5G European Validation platform for Extensive trials

Deliverable D5.5
Performance diagnosis mechanism and reporting methodology document

Project Details

Call	H2020-ICT-17-2018
Type of Action	RIA
Project start date	01/07/2018
Duration	36 months
GA No	815074

Deliverable Details

Deliverable WP:	WP5
Deliverable Task:	Task T5.4
Deliverable Identifier:	5GEVE_D5.5
Deliverable Title:	Performance diagnosis mechanism and reporting methodology document
Editor(s):	Christos Ntogkas, Evangelos Kosmatos (WINGS)
Author(s):	Elian Kraja, Leonardo Agueci, and Giada Landi (NXW), Arturo Azcorra-Salona, Gines Garcia-Aviles, Jaime Garcia-Reinoso, Luis Felix Gonzalez-Blazquez (UC3M), Christos Ntogkas, Evangelos Kosmatos, Panagiotis Demestichas, Vera Stavroulaki, Nelly Giannopoulou, Ioannis Belikaidis, Vassilis Foteinos, Thanos Gkiolias (WINGS)
Reviewer(s):	Giada Landi (NXW), Claudio Casetti (CNIT), Juan Rodriguez (TID), Aitor Zabala (TELC)
Contractual Date of Delivery:	30/06/2020
Submission Date:	26/06/2020
Dissemination Level:	PU
Status:	Final
Version:	V1.0
File Name:	5GEVE_D5.5_v1.0

Table of Contents

List of Acronyms and Abbreviations

Acronym	*Meaning*
5G-PPP	5G Infrastructure Public Private Partnership
API	Application Programming Interface
BMU	Best Matching Unit
CtxB	Context Blueprint
DN	Data Network
E2E	End 2 End
EEM	Experiment Execution Manager
ELM	Experiment Lifecycle Manager
eMBB	Enhanced Mobile Broadband
EPC	Evolved Packet Core
ExpB	Experiment Blueprint
FSM	Finite State Machine
gNB	Next-generation NodeB
GUI	Graphical User Interface
ICMP	Internet Control Message Protocol
IP	Internet Protocol
ITU	International Telecommunications Union
IWF	Interworking Framework
KPI	Key Performance Indicator
MAC	Medium Access Control
mMTC	Massive Machine Type Communication
NG-RAN	Next Generation Radio Access Network
OTT	One-Trip Time
PD	Performance Diagnosis
QoS	Quality of Service
RAN	Radio Access Network
RAV	Results Analysis and Validation
RC	Runtime Configuration
RCA	Root Cause Analysis
REST	Representational State Transfer
RTT	Round-Trip Time
SLA	Service Level Agreement
SOM	Self-Organizing Maps
TCB	Testcase Blueprint

UC	Use Case
UE	User Equipment
URLLC	Ultra-Reliable Low Latency Communications
VNF	Virtual Network Function
VR	Virtual Reality
VSB	Vertical Service Blueprint

List of Figures

List of Tables

Executive Summary

The main objective of 5G EVE WP5 is the design and development of a testing and validation framework as part of the 5G EVE platform. The details of the testing and validation methodology as well as the different components implemented in order to realise this are presented in detail in D5.2 [1].

In the testing and validation process, two features are of paramount importance for a practical and useful platform for the experimenters willing to run their experiments on top of 5G EVE infrastructure: a) the generation of a complete and instructive validation report; b) the provision of a tool for performance diagnosis and root cause analysis.

Regarding the first aspect, the development of a complete and well-defined results analysis and validation report is of high importance because it provides experimenters with clear answers to their initial validation questions. In addition, this report gives a complete picture of the testing and validation process realised using the 5G EVE platform including, in addition to passed/failed results, information about the actual testing process, the environment in which the tests are executed, the different test cases executed to provide the results, a set of high level configuration information. All this information can be provided in a way that will ensure the transparency and repeatability of the entire testing and benchmarking process.

This document presents the details of the approach adopted to build the experiment validation reports. According to the approach the final report can be consolidated in a composition of 4 separate reports created by the elements of the EVE Platform. This happens because, while the main focus of the experiments is the KPI validation, the information regarding the site, the conditions and the technologies used can be extremely useful and insightful to the experimenter as well.

Regarding the second aspect, 5G EVE extends the functionalities of the testing and validation process described in D5.2 [1] with performance diagnosis functionalities. The main objectives of the developed performance diagnosis mechanism are: a) to provide insights on the execution and validation of the tests, b) to execute diagnostics and root cause analysis mechanisms in case of performance degradations.

In this document the performance diagnosis mechanism designed and developed in the project is presented, with their main functionalities including to perform a deeper KPI analysis, to execute anomaly detection on the collected metrics, to perform Root Cause Analysis (RCA) and finally generate a set of proposed actions for alleviating any performance degradation. In addition, initial results from the application of the performance diagnosis mechanisms on emulated networks are presented and analysed.

1 Introduction

Deliverable D5.5 presents the 5G EVE reporting methodology and performance diagnosis mechanisms designed and developed in the project, which are part of the 5G EVE testing and validation platform.

1.1 Overall 5G EVE approach

The main global service provided by the 5G EVE platform for external experimenters is actually the Testing and Validation capability. The high-level workflow for a validation process, as defined in WP1, consists of the four phases presented in Figure 1.

Figure 1: 5G EVE high-level validation process workflow

During the Test Design phase, the direct action and inputs from experimenters are critical. In this phase, the experimentation goals and expected results are jointly defined, together with the test external conditions and assumptions. This, together with the information about what components the Vertical is bringing to the experiment, generates a test case specification to which the 5G EVE consortium can execute a feasibility analysis.

If passed, the test case specification which defines the experiments at the experimenter-level must be translated into a more technical and execution-oriented language. In other words, a meaningful test plan needs to be generated, including not only the inputs from the experimenters among others, but also the testing procedures, the test configurations, the measurable KPIs (vertical- and network-related), and the required supporting infrastructure, etc. The final outcome of this phase is a planning handshake, which will determine the time frame when the experimenter will be able to conduct its tests.

During the execution phase, all the experiment components are instantiated and configured, all the configurations are put in place and, in summary, the infrastructure is effectively adapted to the requirements of the test plan. Only after every component is ready and correctly configured the experiment will be able to start. In line with that, the experiment and the network conditions are continuously monitored, to be able to correlate during the analysis phase the final results with the conditions at each moment.

Finally, results are collected and analysed, and a final report is generated. There are many levels of analysis, depending on many occasions on the KPIs which are meaningful for each specific Vertical. Network KPIs will be measured by default by the 5G EVE platform, in order to validate the results against the 5G PPP KPIs and as a way to ensure that the obtained results are valid (i.e. obtained under correct network conditions).

This deliverable describes in detail two aspects of the final “Test Evaluation and Results Analysis” phase, which are:

- The reporting methodology framework, which deals with how the final results (including test realisation information, network deployment information, metrics and KPIs related information) are presented to the vertical users.
- The performance diagnosis framework, which provides insights regarding the observed performance of the experiments, while is also capable of providing suggestions on performance improvements, by applying post-process analytics on the collected KPIs and metrics.

1.2 Document objectives

The main objectives of this document (and of Task 5.4), as also described in the project's DoW, are twofold. The first objective is to *maximize the impact of the testing and benchmarking procedures by designing and implementing an advanced performance diagnosis mechanism based on enhanced data analytics processes*. In this direction, this document describes in detail the following aspects of performance diagnosis:

- The methodology that the performance diagnosis mechanism will follow.
- The architectural design of the performance diagnosis mechanism and the related functionalities (either in WP5 or in other WPs) required for the performance diagnosis module.
- The development of the performance diagnosis mechanism based, when possible, on open solutions.
- The design and development of the performance diagnosis and Root Cause Analysis (RCA) algorithms, which provide the intelligence of the mechanisms.
- The presentation of initial results from the application of the performance diagnosis mechanism on an emulated network.

The second objective of this document (and of Task 5.4) is *the development of a methodology for concise and clear reporting which will be implemented throughout this work package and will ensure the transparency and repeatability of the entire testing and benchmarking process. The reporting methodology will ensure the detailed documentation of the design process of the testing and benchmarking tools as well as the test case and scenario aspects, while it will also provide an extensive account of the results per technology and test case and the conclusions drawn in each case.* In this direction, this document describes the following aspects of the reporting methodology:

- The description of the adopted methodology for the reporting of the test results
- The description of the different parts of the test result report including: the test scheduling part, the test execution part, the test case validation part
- The reporting workflows and APIs presenting the interaction with the other 5G EVE components
- The presentation of indicative reporting results from the application of the reporting methodology on different use cases.

1.3 Document structure

In addition to the current section, which presents an introduction and the objectives of the document, in this sub-section of the document we briefly introduce the remaining three sections.

In Section 2, the reporting framework is presented. Initially, in section 2, the adopted methodology is described including the list of network KPIs that will be validated and reported. Then, the workflow and APIs are illustrated, focusing on the main components that interact with the reporting framework, i.e. the Experiment Execution Manager and the Results Analysis and Validation components. Finally, in Section 2 indicative reporting results from the application of the reporting methodology on different use cases are presented.

In Section 3, the performance diagnosis mechanism is presented. Initially the general approach adopted in 5G EVE for the design of the performance diagnosis mechanism is presented. Here, the various steps of a performance diagnosis methodology are presented, providing details for each step and justifying which of them are inside the scope of the project. Then, the workflow and APIs are presented in order to stress the interaction with the other 5G EVE components. The design and development of performance diagnosis and root cause analysis algorithms are the heart of the mechanism. Therefore, the adopted algorithms are presented in Section 3 as well. Finally, initial results from the application of the performance diagnosis on emulated networks are presented and analysed.

Finally, section 4 includes the conclusions and next steps of WP5 work.

2 Reporting framework

In all stages of the project's platform operation various functionalities have been implemented in order to provide detailed information regarding the actions of the experimenter and the status and the results of any experiments that have been requested. These independent functionalities are part of the reporting framework.

The goal of the experiments performed using the platform is to validate network KPIs as well as application KPIs provided by the verticals. Based on that, it might be of interest to the experimenter to know, either by documentation or by including that information in the report, how the platform is handling the validation of these KPIs.

5G EVE project is clearly defining and providing a detailed description of the methodology (procedure and steps) followed by the supported 5G-PPP network KPIs verification and validation process. Table 1 summarizes these KPIs along with the minimum requirements. Additionally, we identify software tools that can be used to provide the initial KPI measurements for an end-to-end system or service in an automated way.

The first deliverable of WP5, D5.1, provided a general methodology, tools and measurements for bandwidth and latency metrics. The first approach was conducted within WP1 and its related deliverables, however, our intention is to cover also the specific KPIs that are having a greater impact in our verification procedure.

Table 1: 5G EVE list of Network KPIs

5G EVE KPIs	ITU-R M.24010-0 (11/2017)	Minimum requirements
Speed	DL User Experienced Data Rate (Mbps)	100 Mbps
	UL User Experienced Data Rate (Mbps)	50 Mbps
Capacity	Area Traffic Capacity (Mbit/s/m2)	10 Mbit/s/m2
Latency	User Plane Latency (ms)	1ms (URLLC), 4ms (eMBB)
	Control Plane Latency (ms)	<20ms
Device Density	Connection Density (devices/km2)	1 M devices/ km2 (mMTC)
Mobility	Stationary (km/h)	0 km/h
	Pedestrian (km/h)	0 km/h to 10 km/h

The network KPIs [2] in which this deliverable is focusing on are:

1. **Speed – Experienced user throughput (Mbps).** Specified as bi-directional data throughput, Downlink (DL) and Uplink (UL), that an end-user is able to achieve on the user plane, preferably on MAC layer avoiding the overhead of the upper layers. This KPI is one of the dominant measurements for the Quality of Experience (QoE) level that a user experiences for an active application/service.
2. **Capacity – Area Traffic Capacity (Mbit/s/m^2).** Specified as total traffic throughput served per geographical area. This KPI is also defined as **Traffic Volume Density**, describing the total user data volume transferred to/from end-user devices during a predefined time span divided by the area size covered by the radio nodes belonging to the RAN(s).
3. **Latency – User Plane & Control Plane (ms).** For user plane latency, we can consider two types of metrics, end-to-end latency or one-trip time (OTT) latency and round-trip time (RTT) latency. Ideally, latency should be measured in the MAC layer, not accumulating any processing time on higher layers due to the used application. On the other hand, control plane latency is defined by ITU as the transition time from a most "battery efficient" state (e.g. Idle state) to the start of continuous data transfer (e.g. Active State). This requirement is defined for the purpose of evaluation in the eMBB and URLLC usage scenarios.. An illustration of control plane signalling during transition from Inactive to Active states is shown in Figure 2.

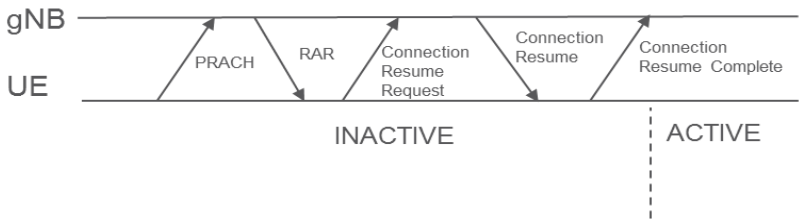


Figure 2: Control plane signalling during transition from Inactive to Active states

4. **Device Density – Connection Density (devices/km^2).** Specified as the total number of simultaneous active connections per square kilometres fulfilling a target QoS, where "active" means the devices are exchanging data within the network. Number of connected/accessible devices are available in NG-RAN node.

5. **Mobility – Stationary / Pedestrian (km/h).** Specified as the system's ability to provide seamless service experience to users that are moving. Two modes are considered in the frames of 5G-EVE project. "Stationary" having a speed of 0 km/h and "Pedestrian" having a speed of 0 to 10 km/h. Seamless service experience means that UE achieves QoS required by the service/application.

While the main focus of the experiments is the validation of the aforementioned KPIs with the selected processes and tools along with any application KPIs, the information regarding the site, the conditions and the technologies used can be extremely useful and insightful to the experimenter as well.

Reporting the information generated from each of the various stages of the experiment definition, preparation and execution processes happen independently in the respective stages. This information can be consolidated in a composition of 4 separate reports created by the elements of the EVE Platform, as seen in Figure 3. These are:

- Test Case Validation report: generated by the RAV module. It pertains to the targeted KPI that the Vertical wants to validate containing information regarding the behaviour of the KPI throughout the test run as well as the final result of the validation process.
- Test Case report: generated by the EEM. It includes the results of the experiment operations returned by the Runtime Configurator (RC) and it is an operational report mainly focused on the different stages of the test execution process and less about the KPIs.
- Experiment report: generated by the ELM. This report contains all the information regarding the requested parameters, technologies used, use case details and other details pertaining to the experiment at a higher level.
- Scheduling report: generated by the Portal. The information produced by the Portal, as the name suggests, are all related to scheduling experiment executions like the time slot, the one or more selected sites and so on.

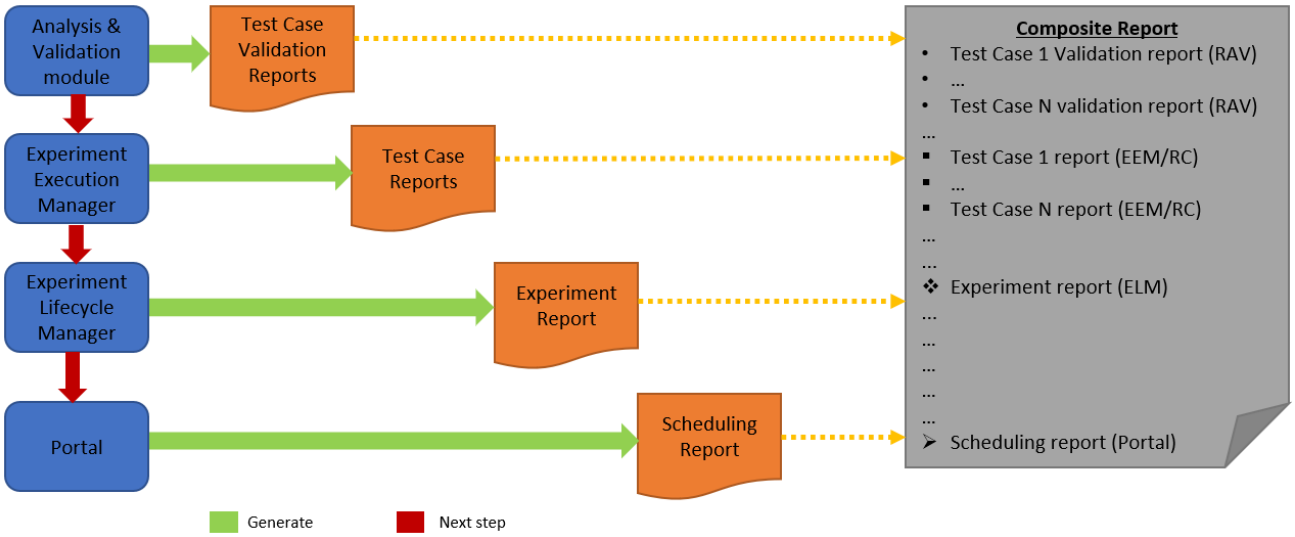


Figure 3: Composite report

The generated reports, especially the Test Case Validation reports and the Test Case reports, besides being useful to the experimenter, will also provide the necessary input for the Performance Diagnosis. Having the detailed reports of multiple test case runs available beforehand can speed up the process and aid in increasing the accuracy of the performance diagnosis as well as the impact of the performance improvement suggestions.

2.1 Reporting framework workflows and APIs

The reporting framework described in this deliverable showcases the form of the final report that the vertical will receive. Currently the reported information during the stages of definition and scheduling of the experiment is provided in the respective steps of the interaction with the platform. After the experiment has begun its execution, the experiment report begins to be dynamically generated after each test case is completed. The workflow of this process can be seen in Figure 4.

After each testcase is finished, the results for that testcase are generated. When all the testcases of the experiment have completed their execution, the results from each testcase is consolidated into the final report. At that point the EEM is notified for the generation of the report which becomes available from the portal.

In order for the final, composite report to include the information regarding all the stages of the experiment, additional communication between existing or currently missing endpoints of the platform elements is required. The proposed modifications to the existing workflow in order to enable the generation of the composite report can be seen in Figure 5.

According to the proposed workflow, after the report for all the testcases is generated, a request for the experiment ID in the upper layers of the platform is sent to the EEM. This experiment ID is a globally unique identifier assigned to the experiment from the experiment definition stage and can be used to correlate all the information surrounding the experiment. Using that ID, the reporting module can request additional information regarding the experiment such as the selected blueprints information, deployed infrastructure and physical and virtual resources information in order to embed that into the composite report as well. This information can be obtained from the ELM and the Multi-Site Inventory. Finally, after inserting this information into the composite report, the Portal is notified that the final report is ready and available to the experimenter.

This extended workflow will provide the experimenter with the composite report containing all the information generated in all stages of the interaction with the platform. Feedback on this report from the first verticals utilizing the 5G EVE platform will be valuable in order to improve aspects like quality of the information, comprehension, appearance, etc.

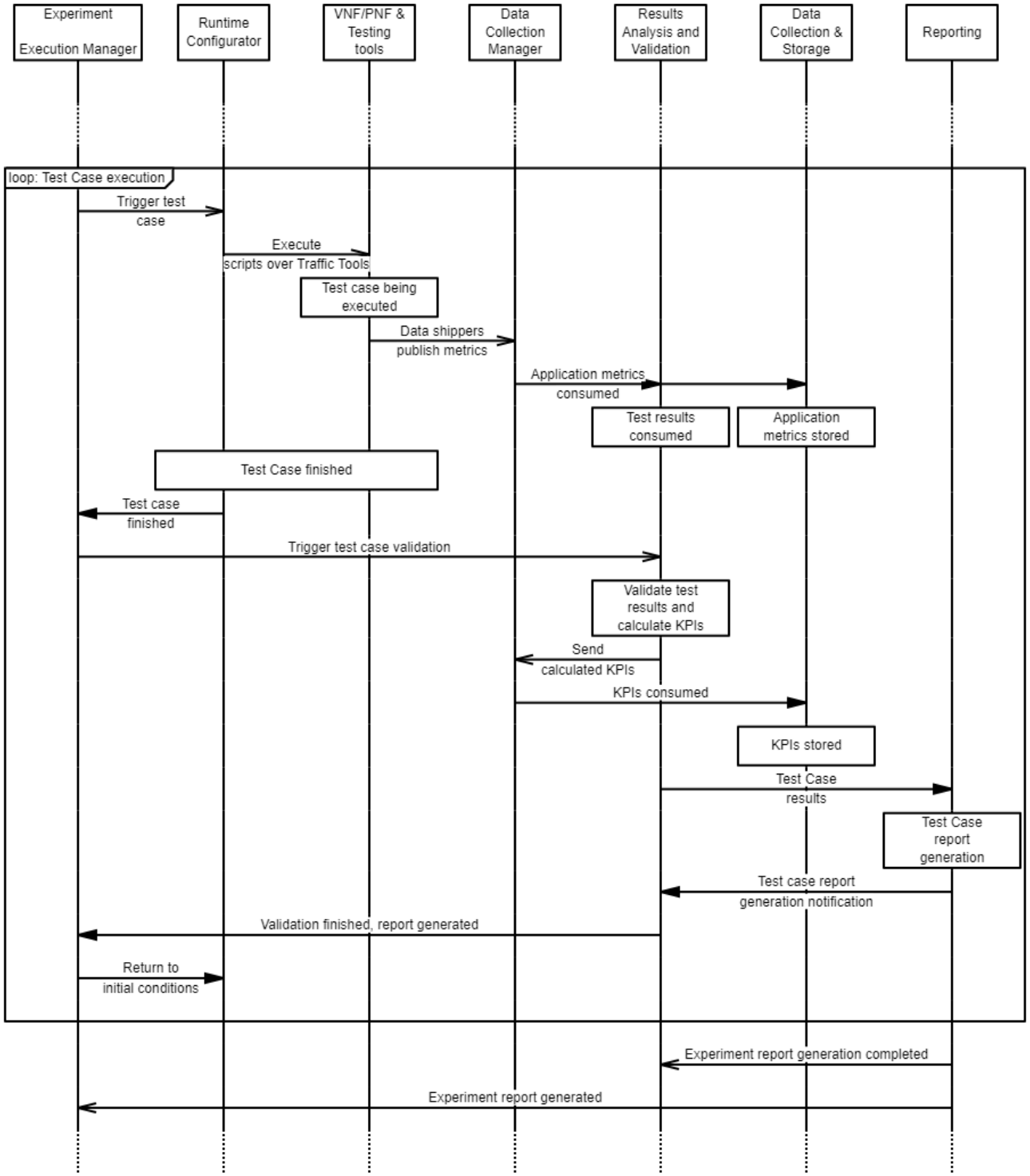


Figure 4: Reporting workflow

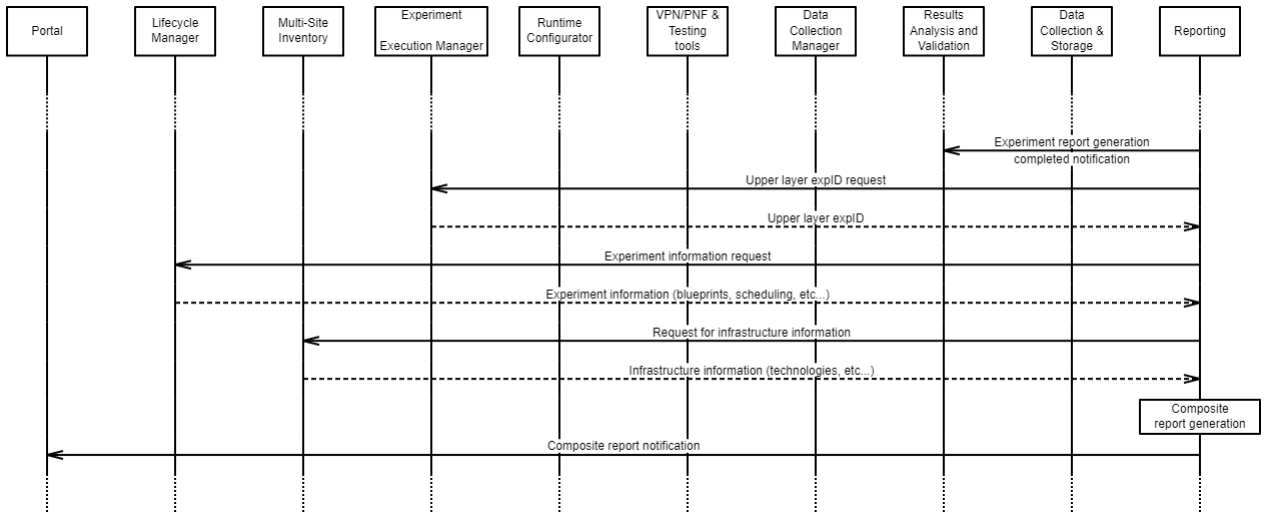


Figure 5: Extended reporting workflow

2.2 Reporting framework components

The components of the reporting framework and their individual reports are examined in the order of the experimenter's interaction with the platform. After the initial phase of designing the experiment comes the scheduling phase. The experimenter can request a time slot to run the experiment in one or more of the available sites compatible with the experiment specifications. The experimenter can monitor and control, to a degree based on service and site specifications, all the experiments requested and executed by him using the corresponding portal dashboard reporting their status, as shown in Figure 6. During the scheduling phase, the experimenter can monitor the status of the scheduling requests for the requested experiments, using the Descriptor ID, along with information regarding the selected site and the execution status if the experiment is already running.

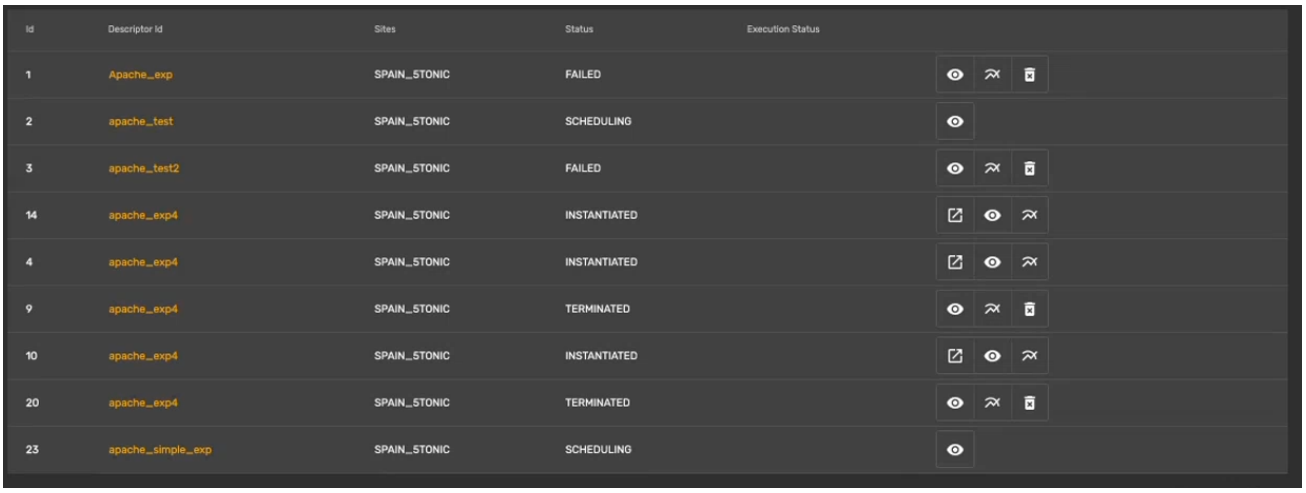


Figure 6: Experiment scheduling status

More in-depth details regarding the experiment can also be accessed through the Experiment details report available for the designed experiments, shown in Figure 7. Such details include the selected site the experiment is run on, the descriptors that were used in the design phase of the experiment, the selected test case as well as links to the report of the experiment and the specific test case results. This information can be useful to the Vertical in order to fine-tune aspects of the experiment thus producing more valuable results for the specific use case.

Field	Value
Name	apache_exec9
Status	RUNNING_EXECUTION
Experiment Descriptor	apache_exp
Target Sites	SPAIN_5TONIC
Target Use Case	Ares2T, apache
Time Slot	Start Date: 01/01/1970, 01:33:40 Stop Date: 01/01/1970, 01:33:40

Execution	State	Report	Test Case Results
apache_exec3	RUNNING		

Figure 7: Experiment details

Additional technical information such as used technologies and service specifics, seen in Figure 8, are also available and can be included in the report through the interaction with other elements of the platform, like the Multi-Site Inventory and the Multi-Site Catalogue.

Field	Value
Name	Simple VSB with Apache
Version	1.0
Description	Simple VSB with an Apache instance and a request generator.
Parameters	Number of tracked devices
Components	apache ueab
Endpoints	cp_apache_int cp_ueab_int cp_ueab_ext sap_ueab
Context Blueprints	
Compatible Sites	SPAIN_5TONIC

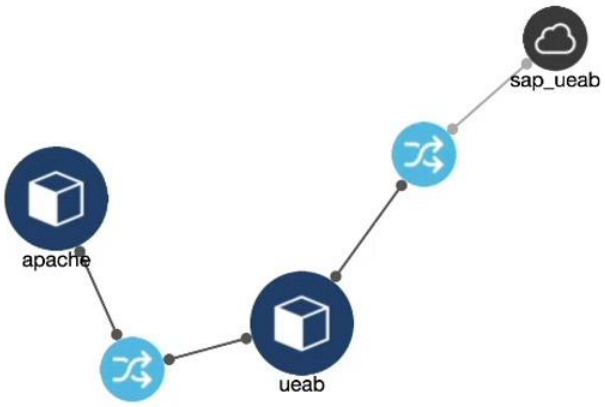


Figure 8: Vertical service information

The information generated from these individual components will be consolidated in the final composite report, along with the validation results, which are described in the next sub-chapter, generated during the execution of the experiment in order to provide a complete E2E report of the experiment.

After an experiment is instantiated, the next components that generate meaningful reports for the experimenter are the EEM and the RC. The EEM handles the high-level test case execution and produces a report that focuses on the operational status and results of each test case, shown in Figure 9 of the experiment execution.

Execution	State	Report	Test Case Results
apache_exec45	COMPLETED		Test Case apache_exp47-tc-0
apache_exec47	COMPLETED		Test Case apache_exp47-tc-0

Figure 9: Test case status report

The status of each test case, pass or fail codified by the green or red colour of the test case, is derived from a detailed output generated by the results of the RC's actions. A link to the report is available as well. The RC also handles the Day-2 configuration of the elements and the execution of the specific actions and events that take place during the test case scenario runtime. The report generated from this element is the output of the Ansible playbook[1], an example of which is shown in Figure 10, executing the test case specific actions. This is a more technical report useful for debugging purposes of the experiment and possibly to correlate test case specific actions with fluctuations of the perceived performance using the performance diagnosis module.

```
ok: [tinyenid]
ok: [ahfang]

TASK: [Add repository] ***********************************************************
ok: [halob]
ok: [bobnit]
ok: [ahfang]
ok: [tinyenid]

TASK: [Install packages] **********************************************************
ok: [halob] => (item=btsync,debconf-utils,btsync-gui,transmission-daemon)
ok: [tinyenid] => (item=btsync,debconf-utils,btsync-gui,transmission-daemon)
ok: [ahfang] => (item=btsync,debconf-utils,btsync-gui,transmission-daemon)
ok: [bobnit] => (item=btsync,debconf-utils,btsync-gui,transmission-daemon)

TASK: [Configure BT Sync] *********************************************************
ok: [halob]
ok: [bobnit]
ok: [ahfang]
ok: [tinyenid]

TASK: [Regenerate BT Sync's config file] ******************************************
changed: [halob]
changed: [bobnit]
changed: [ahfang]
changed: [tinyenid]

TASK: [Restart BT Sync service] ***************************************************
changed: [halob]
changed: [ahfang]
changed: [bobnit]
changed: [tinyenid]

PLAY RECAP ************************************************************************
ahfang                     : ok=6    changed=2    unreachable=0    failed=0
bobnit                     : ok=6    changed=2    unreachable=0    failed=0
halob                      : ok=6    changed=2    unreachable=0    failed=0
tinyenid                   : ok=6    changed=2    unreachable=0    failed=0
```

Figure 10: Ansible playbook output from Runtime Configurator

As mentioned earlier, the reports generated from the EEM and the RC are more operational and suitable as debugging reports useful in determining what might have caused an experiment to fail or, combined with the performance diagnosis report and correlating with the KPIs, to even identify which actions led to performance degradation of the service.

2.2.1 Result Analysis and Validation (RAV)

After the experiment's execution has terminated, the Results Analysis and Validation (RAV) of the generated metrics takes place and the selected KPIs are computed and validated. The report generated from the RAV module, which can be seen in **Error! Reference source not found.**, is experimenter specific. It provides information for all the experiments owned by the specific experimenter, with information regarding the time slot, the site they were deployed and test case specifics, like validated KPIs and validation conditions. An experiment is reported successful when all test cases have succeeded and respectively a test case is considered successful when all KPIs for that test case are validated.

[1] https://docs.ansible.com/ansible/latest/user_guide/playbooks.html

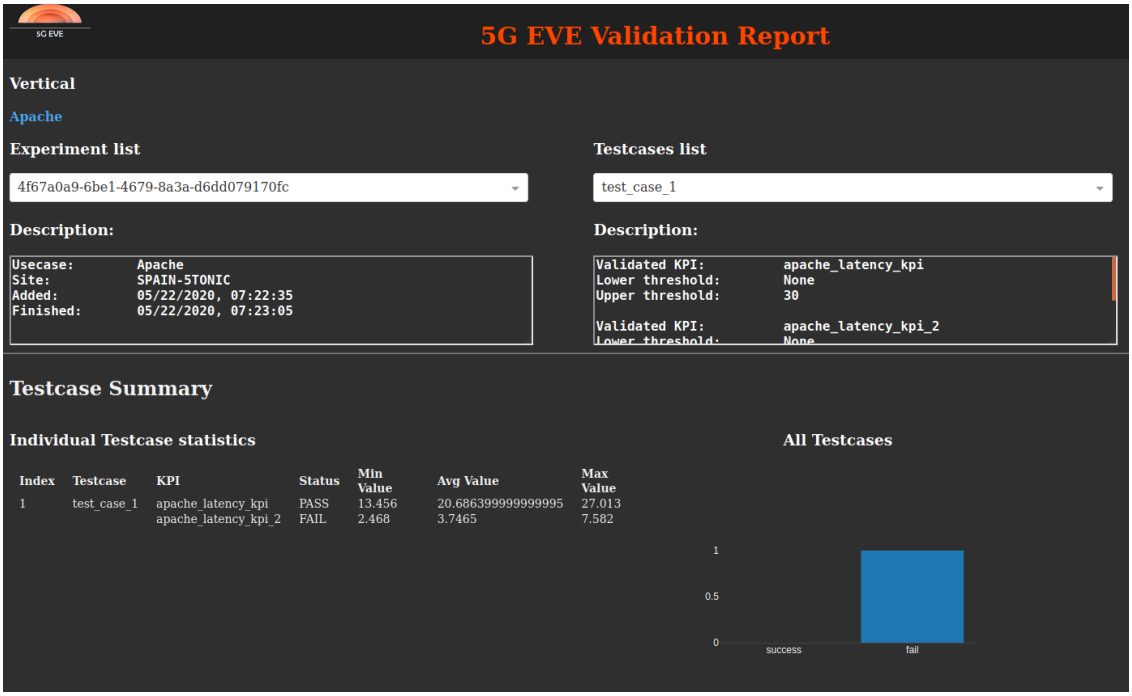


Figure 11: Experiment and test case validation report

A summary of all the test cases executed and the KPIs validated is provided with aggregated statistics for the KPIs including minimum, average and maximum values for each KPI during the course of the test case execution. The test case is marked as failed because one of the KPIs failed to validate. The detailed behaviour of each KPI is also displayed in the corresponding graphs along with the validation conditions. The validation conditions for each KPI set an acceptable range inside which the Vertical's requirements are satisfied. In the showcased example the KPI, apache_latency_kpi marked blue, is within the requested bounds, lower bound marked orange and upper bound marked green, and as such the KPI is validated successfully. Examples of the behaviour of the validated KPIs in a test case can be seen in Figure 12**Error! Reference source not found.**.

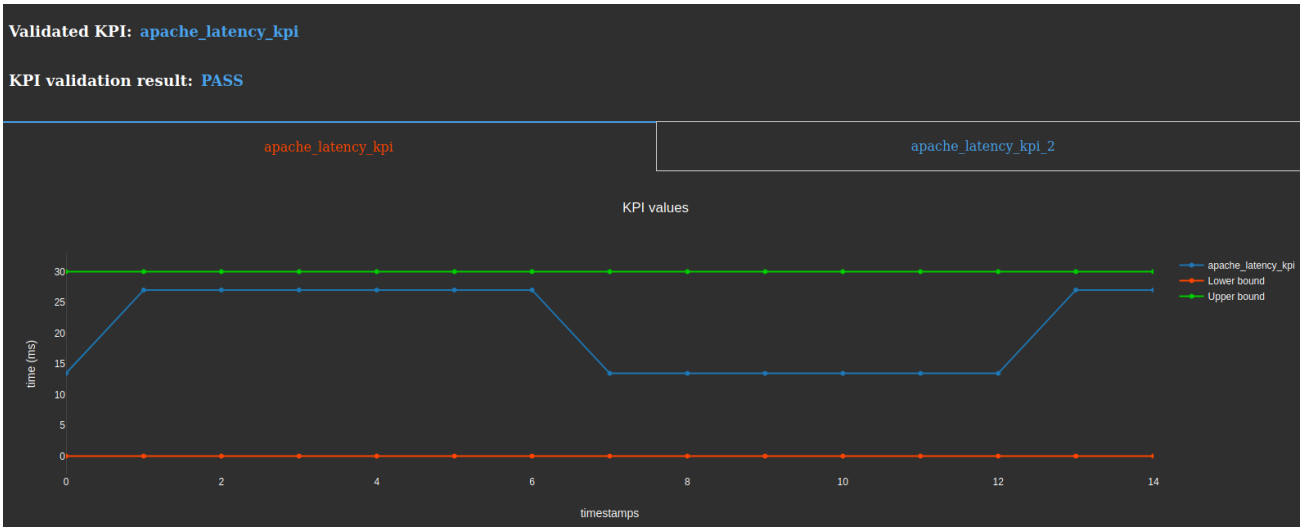


Figure 12: KPI example report #1

As shown in **Error! Reference source not found.**, in single test case multiple KPIs can also be validated. It is very typical for vertical services to have multiple SLAs and as such need to monitor the performance of multiple KPIs at the same time. In this example two KPIs were requested to be validated at the same time. The second KPI, using the temporary name apache_latency_kpi_2, exceeded the requested range of values and so its validation failed.

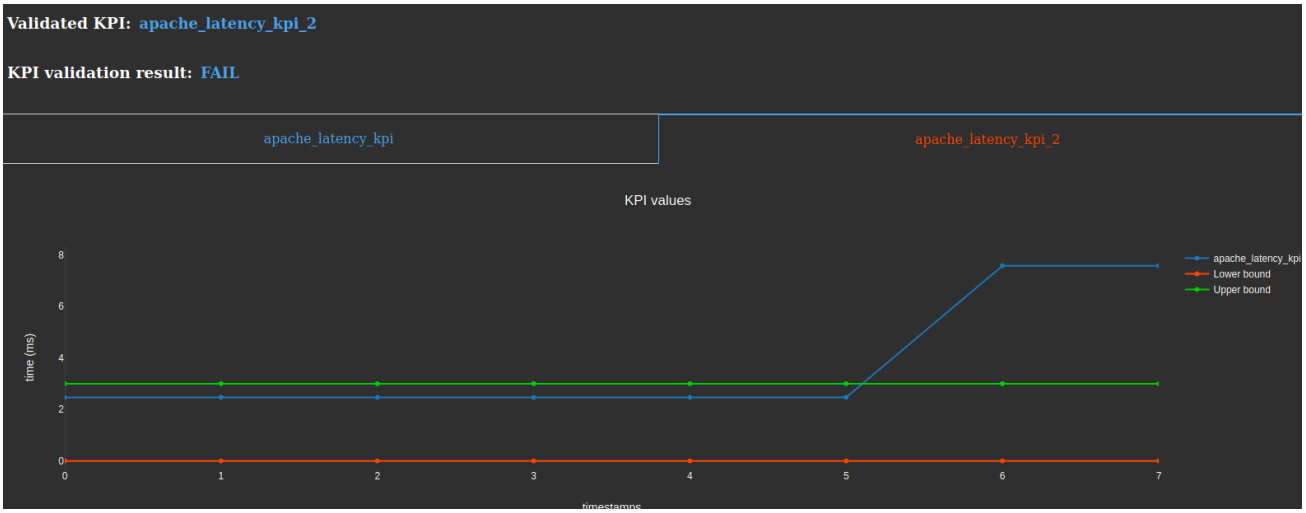


Figure 13: KPI example report #2

The report generated from this module will be further enhanced with the report generated from the Performance Diagnosis module upon request from the experimenter. The contents of the Performance diagnosis report contain information regarding the behaviour of the KPIs, based on the already collected results, information regarding the detection of performance degradations as well as a Root Cause Analysis (RCA) report. The RCA report includes information regarding the cause of the performance degradation and, given sufficient data, it will also include suggestions to improve the observed performance based on typical network troubleshooting methodologies.

2.2.2 Experiment Execution Manager (EEM)

The Experiment Execution Manager (EEM) is the 5G EVE platform component that coordinates all the procedures for the automated execution and validation of experiments in the 5G EVE infrastructure. The high-level architecture and the software design of the EEM have been already introduced in D5.2 [1] and they are still valid for the latest EEM release.

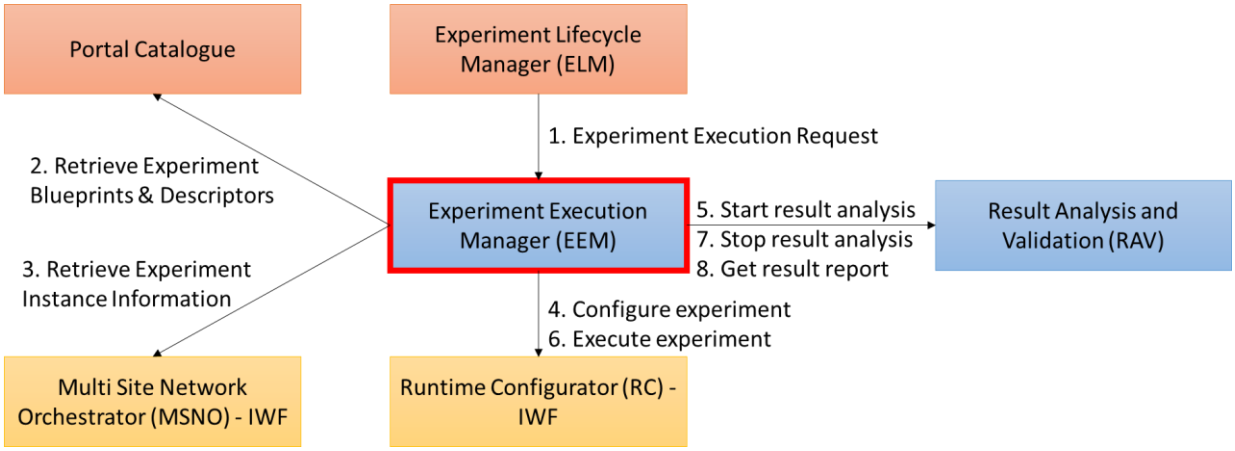


Figure 14: Interaction between Experiment Execution Manager and 5G EVE components

The workflow related to the experiment execution is represented in **Error! Reference source not found.**, showing the interactions between the EEM and the other components of the 5G EVE platform, at the portal and at the Interworking Framework (IWF) level. The overall principles remain the same defined in D5.2, however, it is important to note that some of the interactions have been refined for the MS9 release, requiring the update of the external interfaces and slight changes in the internal finite state machine (FSM) of the EEM engine. The following subsection describes the experiment execution workflow, highlighting the coordination role of the EEM in the entire procedure, and the external interfaces with the other 5G EVE components. In this context, the EEM acts with a server role in the interaction with the Experiment Lifecycle Manager (ELM) and with a client role in all the other communications. In fact, in the former case the EEM receives requests for the

execution of the experiment with a number of test cases, while in the latter case it is the EEM itself that requests information or the execution of actions to the other entities.

2.2.2.1 Workflow for Experiment Execution

The high-level workflow for the execution of an experiment is depicted in **Error! Reference source not found.**, where all the involved entities and their interactions are represented. The experiment execution is part of the lifecycle of an experiment and it can be performed several times once the experiment has been successfully scheduled and deployed on the target 5G EVE facilities. The execution of an experiment is explicitely triggered by an experimenter, who can select a number of test cases and provide different configurations for each of them. This approach allows the experimenter to repeat the execution in different context environments, giving the possibility to compare the related results and KPIs with variable conditions and, thus, supporting the verticals in the tuning of the service for different deployments.

The entire execution and validation procedure is fully automated, following the information defined in the various blueprints. In particular, the vertical service and the context blueprints (VSB and CtxB) specify the application metrics to be collected from the service, while the experiment blueprint (ExpB) specifies the infrastructure metrics to be retrieved from the 5G network and the KPIs to be elaborated processing such metrics.

The experiment descriptor (ExpD) defines threshold-based criteria to evaluate such KPIs and to validate the experiment results. The guidelines for the configuration and the execution of the experiment are provided through the Test Case Blueprints (TCB), which define the scripts to run in the experiment machines with the related variables. Some of these variables refer to the virtual resources instantiated by the system when deploying the experiment (e.g. the IP addresses of the VNFs), while others are configuration parameters that can be specified by the experimenter when requesting the execution. The EEM elaborates all this information and coordinates accordingly the execution of the various actions on the 5G EVE platform.

The detailed execution workflow is described in Table 2, explaining the different steps reported in Figure 14**Error! Reference source not found.**.

Table 2: Experiment execution and validation workflow

Step	Involved entities	Description
1	ELM, EEM	**Experiment execution request.** In this step the ELM sends a request to the EEM for the execution of an experiment that has been deployed in the virtual environment. The request includes the details of the deployed experiment, including the ID of the corrisponding Network Service and information about its blueprints.
2	EEM, Portal Catalogue	**Retrieval of experiment blueprints and descriptors.** In this step the EEM reads the various blueprints and descriptors to get all the information needed to configure, execute and evaluate the experiment. In particular, the EEM retrieves information about the scripts to be launched for the configuration and the execution, their variable parameters, the list of application and infrastructure metrics and KPIs to be collected and evaluated. Guidelines for writing the configuration and execution scripts are available in D3.4 **Error! Reference source not found.**
3	EEM, MSNO	**Retrieval of information on the deployed experiment instance.** In this step the EEM performs a query related to the Network Service instance associated to the experiment. This query allows the EEM to retrieve the VNFs' IP addresses that must be included in the configuration and execution scripts.

4	EEM, RC	**Experiment configuration.** In this step the EEM interacts with the RC to request the Day 2 configuration of the VNFs and the configuration of the data shippers in the site facilities for the collection of infrastructure metrics. In the former case, the EEM sends the configuration script retrieved from the TCB and filled with the runtime parameters provided by the user or gathered from the MSNO. In the latter case the EEM send the list of the infrastructure metrics that must be collected, as specified in the ExpB.
5	EEM, RAV	**Activation of result analysis procedure.** In this step the EEM interacts with the RAV to activate the process of collecting metrics and compute KPIs, providing all the related details available in the blueprints.
6	EEM, RC	**Experiment execution.** In this step the EEM interacts with the RC to request the execution of the script to run the experiment. As in step 4, the EEM retrieves the script from the TCB and completes it with the runtime parameters provided by the user or gathered from the MSNO. After triggering the execution, the EEM starts a polling thread to query the RC about the execution status and to follow the progress of the experiment.
7	EEM, RAV	**Termination of execution and activation of result analysis procedure.** In this step the EEM notifies the RAV about the termination of the experiment execution and requests the RAV to proceed with the result analysis.
8	EEM, RAV	**Query of result report.** In this step the EEM sends a query to the RAV for retrieving the URL of the report generated in the result analysis procedure. This URL is reported to the ELM in the execution record, so that the results can be displayed in the GUI.

2.3 Reporting Use Case results

In preparation for the release of the platform a few use cases have already been used to test the operation of the reporting framework and mainly the RAV results reporting. Some of those use cases along with the generated results are showcased in this section.

2.3.1 Utilities use case

The first use case that was used to test the reporting framework was the Utilities use case from the Greek site. This use case describes a scenario of a smart grid containing multiple voltage and current sensors and smart nodes that manage the grid's interconnections. In the case of a power outage in a node the smart grid management platform decides on the optimal routing of power to that node targeting the least amount of down time. For this use case the KPIs of interest are the following:

- *Power restoration time*: The down time of a node from the moment the node loses power until the moment the power to the node is restored. It is comprised of the latency for the two messages, power outage notification and power restoration decision, the time it takes for the platform to identify the optimal route, the power restoration decision time, and an internal reconfiguration period in order to switch to the new path. This KPI is linearly affected by latency.
- *Power restoration decision time*: The processing time for the platform to decide the optimal route of power for the specific node that is experiencing the power outage.
- *RTT latency*: The roundtrip latency of the messages, in this case measured from a node to the platform and back.

After the requested experiment for this use case has finished, the reporting framework generated the Experiments results report, shown in After the experiment has concluded, the analysis of the generated KPIs demonstrates that the selected infrastructure technologies and element configuration has yielded Power Restoration Decision Time, Power Restoration Time and RTT Latency times acceptable by the Verticals

requested range. As such, the experiment is considered successful. The aggregated statistics of the KPI values fluctuations can also be further analysed in order to provide insights to the behaviour of the KPIs when using other infrastructure options or element configurations..

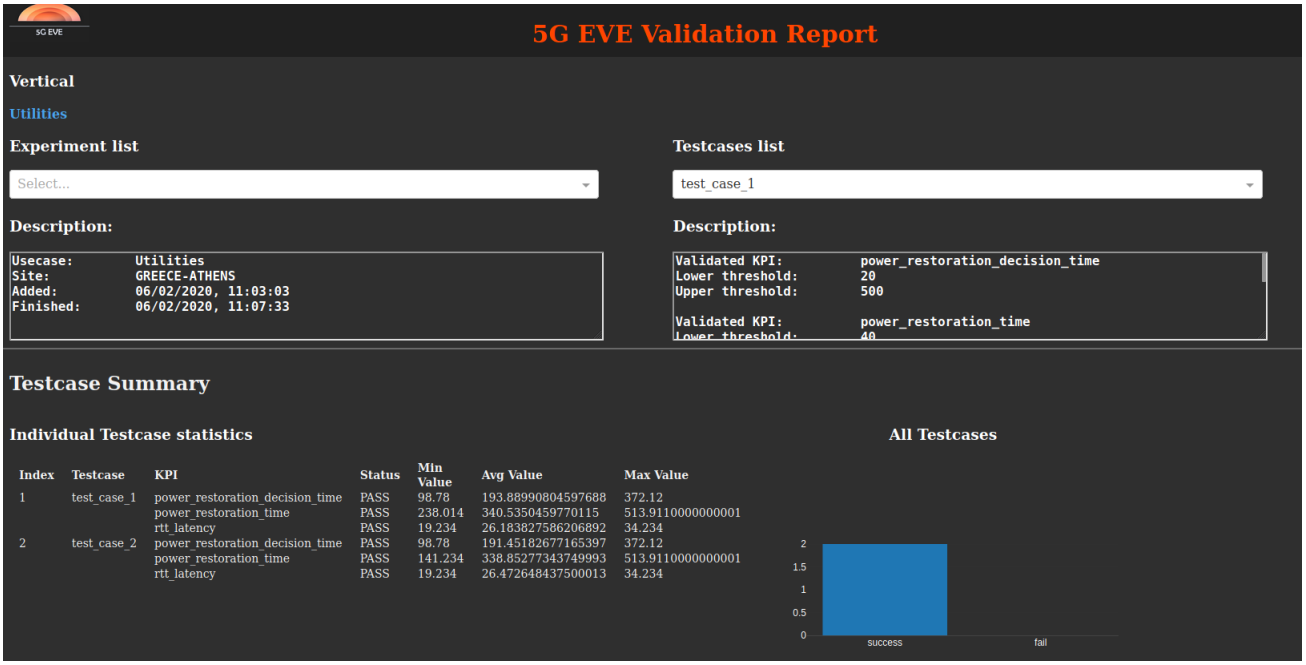


Figure 15: Utilities use case overall results

After the experiment has concluded, the analysis of the generated KPIs demonstrates that the selected infrastructure technologies and element configuration has yielded Power Restoration Decision Time, Power Restoration Time and RTT Latency times acceptable by the Verticals requested range. As such, the experiment is considered successful. The aggregated statistics of the KPI values fluctuations can also be further analysed in order to provide insights to the behaviour of the KPIs when using other infrastructure options or element configurations.

The values of the Power Restoration Decision Time, Power Restoration Time and RTT Latency KPIs can be seen in Figure 16, Figure 17 and Figure 18 respectively.

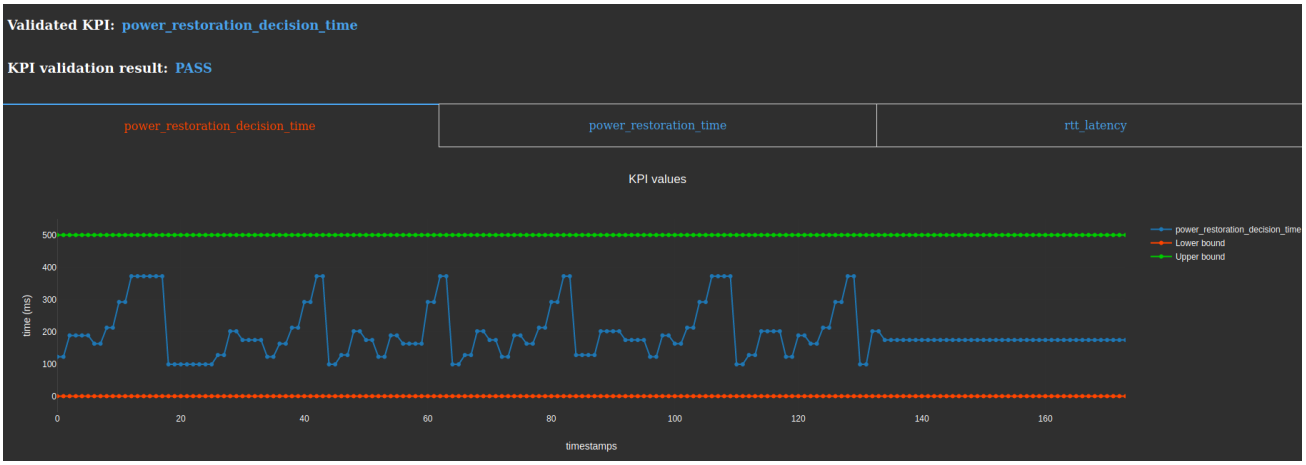


Figure 16: Power restoration decision time KPI

As mentioned earlier, the behaviour of the Power Restoration Decision time appears to have an acceptable fluctuation between the requested margins set by the Vertical. Further analysis can provide insights into which specific actions in the course of the experiment might cause these fluctuations and try to alleviate this behaviour.

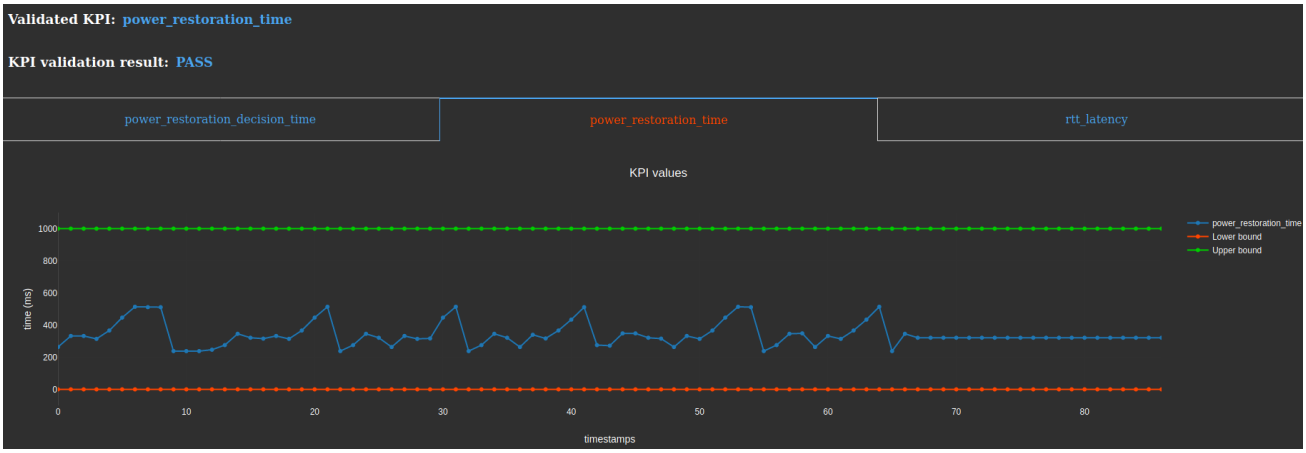


Figure 17: Power restoration time KPI

A likewise behaviour can be observed with the Power Restoration Time KPI, which is to be expected since both this KPI is linearly dependent on the Power Restoration Decision Time and the RTT Latency KPIs.

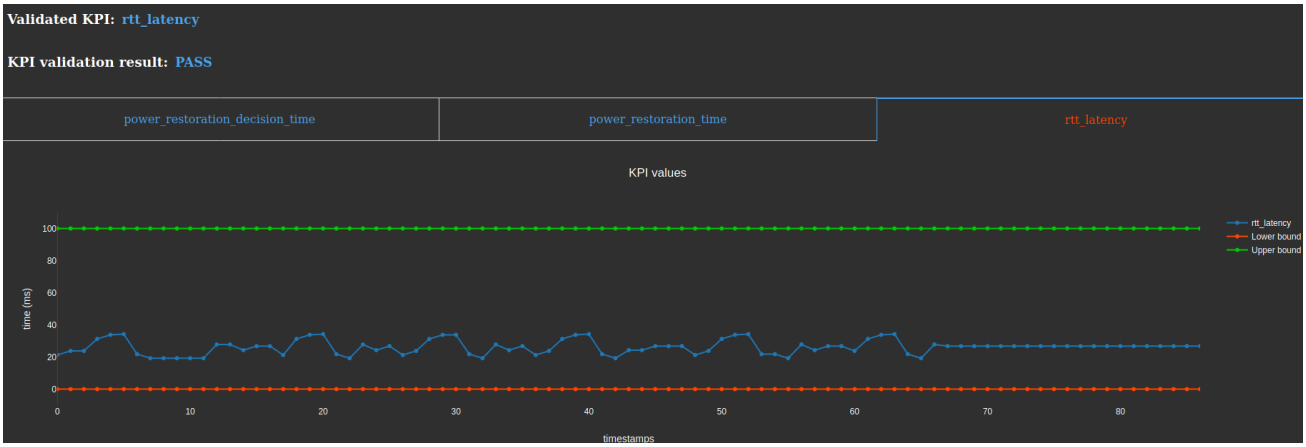


Figure 18: RTT latency KPI

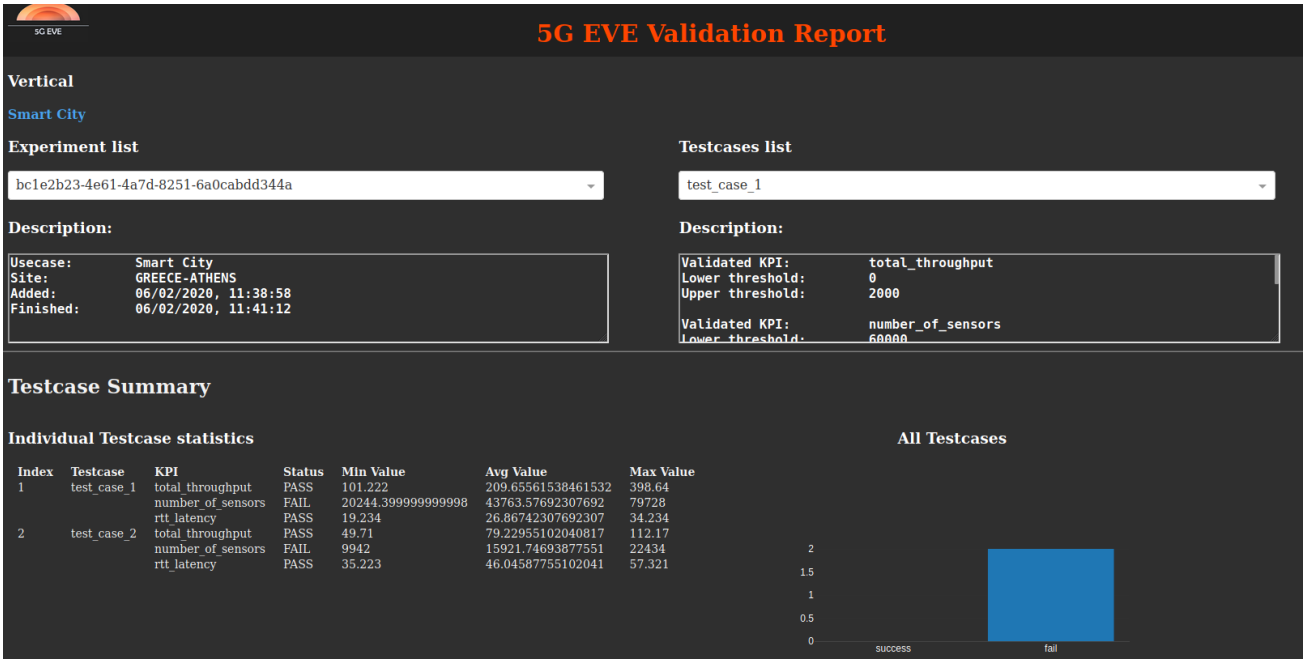
Finally, the RTT Latency is the KPI mostly linked to the infrastructure and technologies deployed in the site and selected for the experiment.

2.3.2 Smart City use case

The second use case that was used to test the reporting framework was the Smart City use case from the Greek site. This use case describes a scenario of a smart city platform monitoring multiple sensors and controlling actuators that provide information and functionalities related to temperature, air quality, water quality and noise monitoring and actuator control. The current setup in the site includes multiple air quality sensors. For this use case the KPIs of interest are the:

- *Total throughput*: The total throughput of the system which depends on the number of sensors transmitting and the required bandwidth.
- *Number of sensors*: The number of sensors that the system is able to accommodate with their bandwidth requirements. Most of the sensors have fixed bandwidth requirements based on the transmission schedule and fixed packets. With that in mind, the number of supported sensors depends on the available bandwidth.
- *RTT latency*: The roundtrip latency of the messages, in this case measured from an air quality sensor to the platform and back.

After the requested experiment for this use case has finished, the reporting framework generated the Experiments results report, shown in Figure 19.



Index	Testcase	KPI	Status	Min Value	Avg Value	Max Value
1	test_case_1	total_throughput	PASS	101.222	209.65561538461532	398.64
		number_of_sensors	FAIL	20244.399999999998	43763.57692307692	79728
		rtt_latency	PASS	19.234	26.86742307692307	34.234
2	test_case_2	total_throughput	PASS	49.71	79.22955102040817	112.17
		number_of_sensors	FAIL	9942	15921.74693877551	22434
		rtt_latency	PASS	35.223	46.04587755102041	57.321

Figure 19: Smart City use case overall results

The values of the KPIs set by the Vertical, Total Throughput, Number of Sensors and RTT Latency, can be seen in After the analysis of the Total Throughput KPI, which is the aggregation of the throughput required for all sensors, during the execution of the experiment, A close examination of the Number of Sensors KPI, though, and The RTT Latency KPI's behaviour is also in accordance to the Total Throughput KPI. Results like these can aid in identifying possible software or hardware implementation limitations that might not be network related. This is a prime example of a use case that would benefit from the Performance Diagnosis module that will be available in the next release of the platform capable of identifying the source of performance degradation or irregular behaviour..

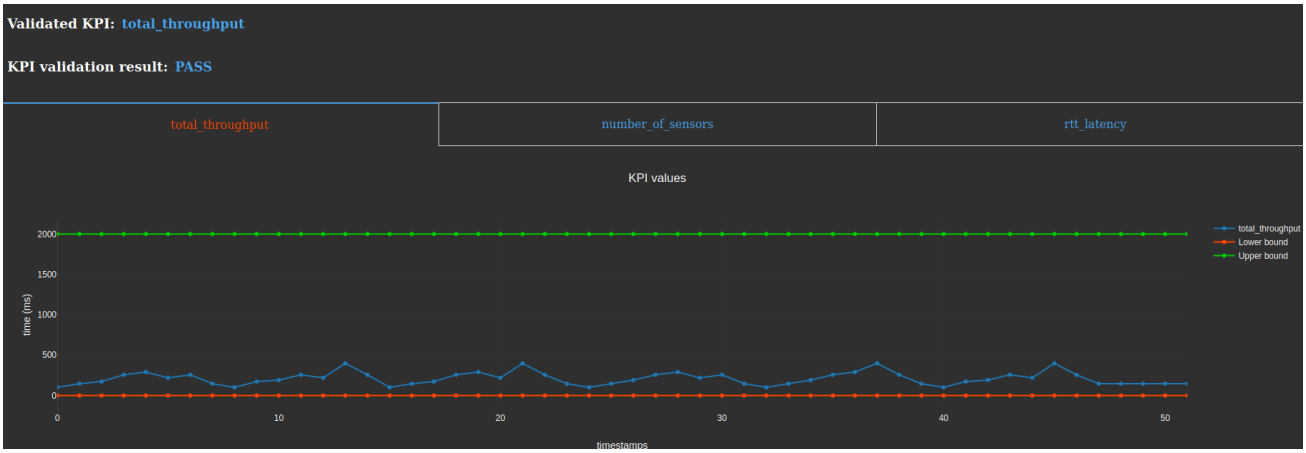


Figure 20: Total throughput KPI

After the analysis of the Total Throughput KPI, which is the aggregation of the throughput required for all sensors, during the execution of the experiment, the requested performance appears to be satisfied with the selected technologies.

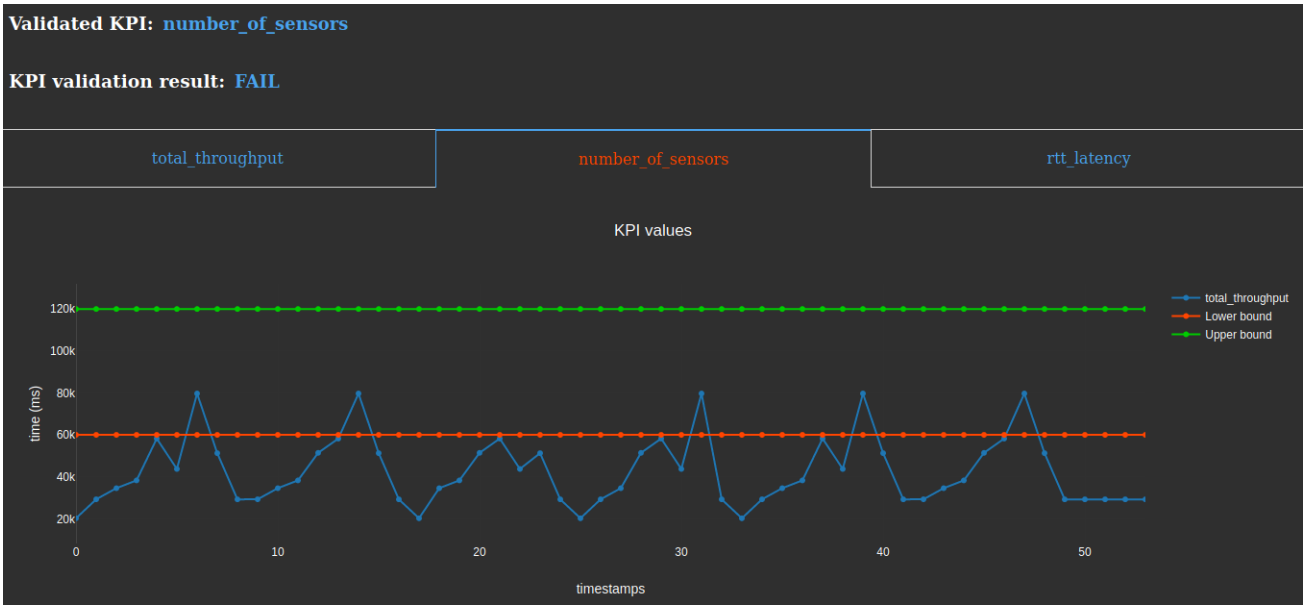


Figure 21: Number of sensors KPI

A close examination of the Number of Sensors KPI, though, tells a different story. It appears that while the throughput is adequate the requested supported number of sensors does not appear to be satisfied in most of the experiment. Synthetic analysis of multiple KPIs can provide insight into the cause of this behaviour. A possible cause for this failure in validation could be software limitations to the Vertical's software or inefficient network or physical resources utilization.

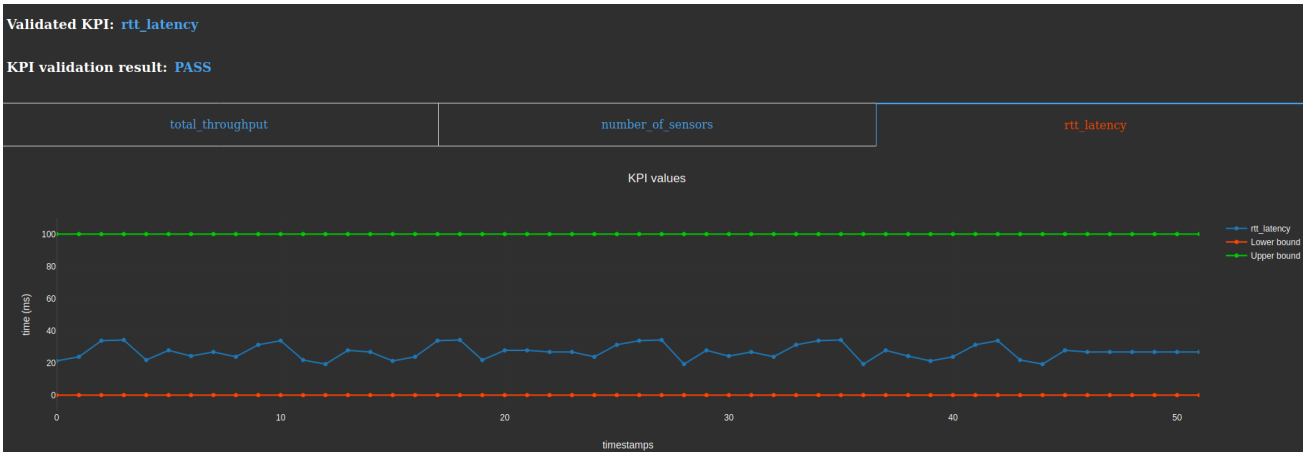


Figure 22: RTT latency KPI

The RTT Latency KPI's behaviour is also in accordance to the Total Throughput KPI. Results like these can aid in identifying possible software or hardware implementation limitations that might not be network related. This is a prime example of a use case that would benefit from the Performance Diagnosis module that will be available in the next release of the platform capable of identifying the source of performance degradation or irregular behaviour.

2.3.3 Tourism use case

The third use case that was used to test the reporting framework was the Tourism use case from the Spanish site. This use case describes a scenario of an event streaming service. Video captured from cameras is sent over the 5G network to a remote server where it is rendered to end-users that enjoy an immersive experience of the event using technologies like VR glasses. The current setup in the site includes a camera, a production and ingest server that sends the video to the remote server and a device implementing the end-user equipment that consumes the video. For this use case the KPIs of interest are the following:

- *User Data Rate in Uplink*: The total throughput of the system depending on the number of cameras transmitting and the required bandwidth.
- *Latency*: The latency between the RTT capturing device and the end-user.

After the requested experiment for this use case has finished, the reporting framework generated the Experiments results report, shown in Figure 23.

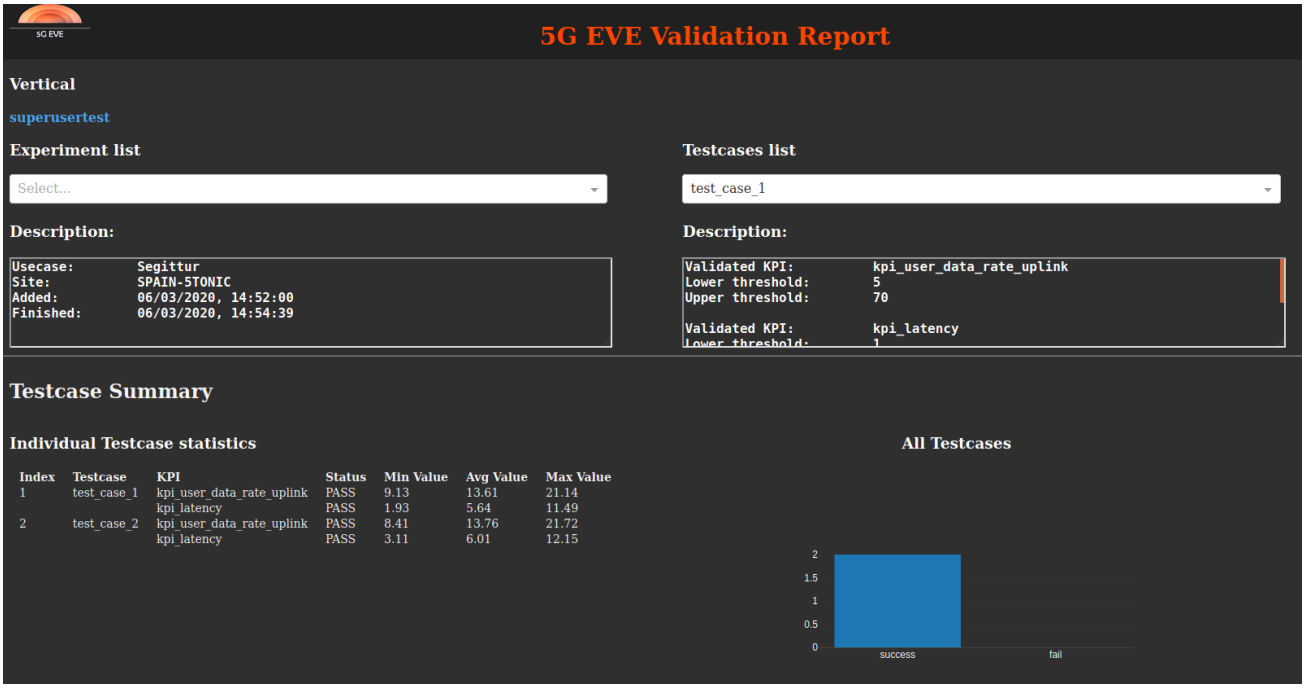


Figure 23: Tourism use case overall results

For the Smart Tourism use case the main focus is the User Data Rate in Uplink and the Latency. The behaviour of these KPIs can be seen in Figure 24 and Figure 25.

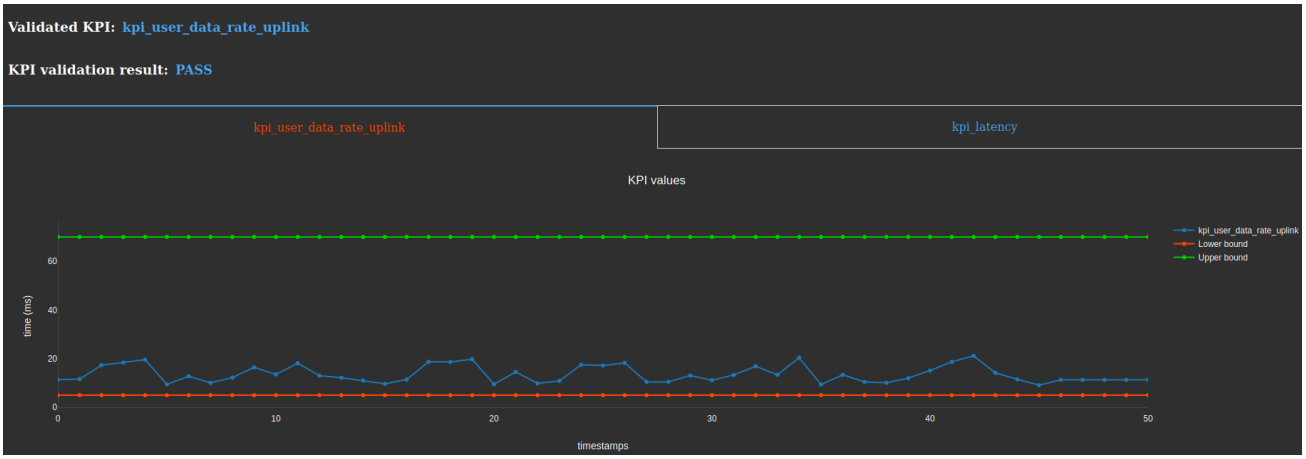


Figure 24: User Data Rate in Uplink KPI

The values of the User Data Rate Uplink while on the lower end of the requested range, are acceptable for the purposes of the use case and so the KPI is validated successfully.

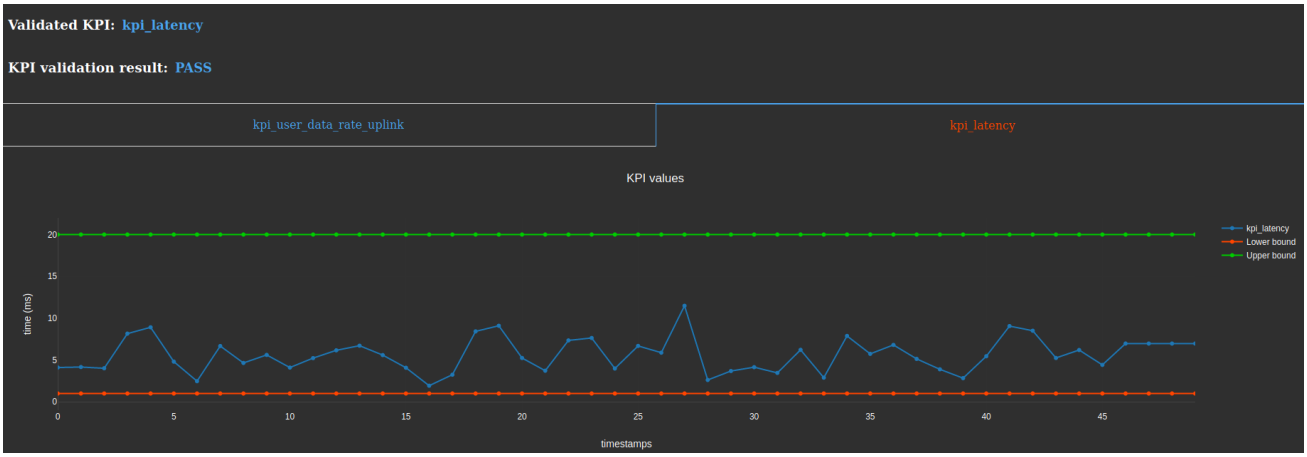


Figure 25: Latency KPI

The Latency requirements for this use case while not strict, are easily satisfied by the underlying technologies, leading to a successful validation of the whole test case scenario.

3 Performance Diagnosis framework

Due to the increased commercial interest in 5G infrastructures there has also been an increase in the interest in software solutions to help deliver reliability in the components of the 5G network, as well as on the services running on top of the 5G network. Therefore, the research into the fields of performance diagnosis and root-cause analysis has been gaining popularity with hopes to find effective methods and models to provide reliability to the 5G services. The hope is that by predicting and localizing faults and service degradations, engineers and technicians can make fact-based decisions on how to improve the system or mitigate the possible faults (**Error! Reference source not found.**,[2],[3]). This in turn would allow for companies to deliver more reliable services **Error! Reference source not found.**. These features, which are critical for the deployment and delivery of 5G services, cannot be absent from a testing and validation platform like 5G EVE.
In order to address the aforementioned challenges, 5G EVE designed and implemented an advanced performance diagnosis mechanism based on enhanced data analytics processes. The performance diagnosis mechanism targets the maximization of the impact of the testing and validation procedures. The developed diagnosis mechanism will offer insights regarding the observed performance and suggesting tips for performance improvement by applying post-process analytics on the collected KPIs.
However, understanding and predicting the performance of a service on the network and on the cloud is by its nature a hard thing to do. The services are often a part of a large and complex software system located in different virtual and physic entities across the 5G network. Therefore, understanding the performance of a system of that magnitude does not only require expert domain knowledge but also analytical models that often tend to be overly complex.

3.1 Performance diagnosis approach

The performance diagnosis approach adopted in 5G EVE is depicted in Figure 26.

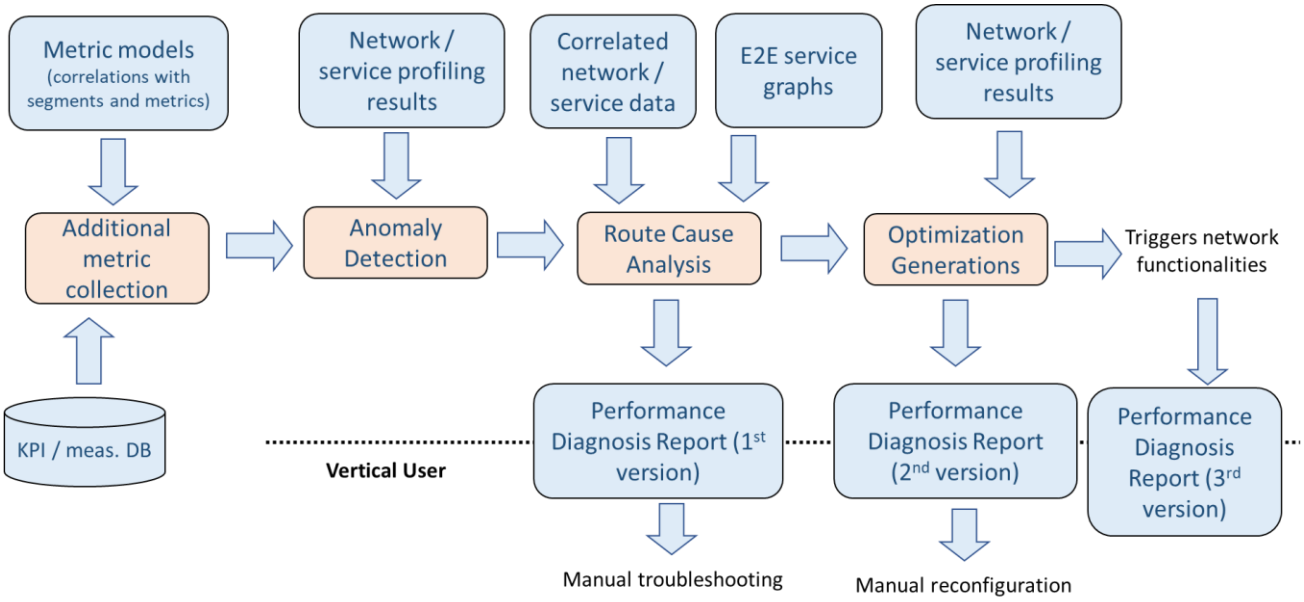


Figure 26: Performance diagnosis approach

The main components of the performance diagnosis mechanism are described below.

Additional metric collection
This module is responsible to collect additional metrics collected during the experiment execution, in addition to the metrics used for the analysis of the experiment KPIs.
These metrics include general network metrics described below:

- Latency

    - Latency between two nodes in the network. Latency can be measured as: a) Round-Trip Time (RTT) latency or; b) One-Trip Time (OTT) latency.
- Throughput metric
    - Throughput measured on an interface of a network node. Throughput can be measured as: a) incoming traffic to the node (ingress throughput) and b) outgoing traffic from the node (egress throughput)
- Node metrics
    - CPU usage (%)
    - MEM usage (%)
    - Disk usage (%)

The aforementioned metrics can become more specific in the case of 5G infrastructure and 5G node characteristics as described below:

- RTT latency
    - Between the UE (client) and the Server (located in the DN/internet) – End to end latency of the whole network
    - Between the UE (client) and a tester PC (connected to the N6 interface) – Latency of the 5G network
    - Between the UE and the EPC
    - Between the UE and the gNB
- Throughput
    - Between the UE (client) and the Server (located in the DN)
    - Between the UE (client) and a Server (connected to the N6 interface)
- Node metrics
    - CPU, MEM, Disk Usage (%) of Server (located in the DN)
    - CPU, MEM, Disk Usage (%) of Server (connected to the N6 interface)
    - CPU, MEM, Disk Usage (%) of 5G CORE VNFs

Finally, the KPI analysis may include metrics coming for the application layer, related with actual operation of the deployed services.

Anomaly detection

Anomaly detection is an important data analysis task that detects anomalous or abnormal data from a given dataset. It is an interesting area of data mining research as it involves discovering enthralling and rare patterns in data. It has been widely studied in statistics and machine learning, and also synonymously termed as outlier detection, novelty detection, deviation detection and exception mining.

The first step of performing diagnosis in a 5G environment after collected all the required metrics (including the additional metrics) is the anomaly detection (**Error! Reference source not found.**,[2]). Before the data can be analysed, and the process of finding the root-cause of a service degradation can be started, the system must first be able to detect that an actual anomaly is present.

In order to identify any anomalies that need to be considered further, the anomaly detection module analyses the collected metrics of the experiments by taking into consideration the set of network profiling results and service profiling results. The network/service profiling results are generated a priori through the process of network and service profiling, and the results are stored for future use by the anomaly detection module. In 5G EVE, the profiles are generated based on the outcomes of a set of experiments with different configuration (network and application) parameters (test cases). The specification of test cases can be found in D5.2 [1].

Root Cause Analysis

The core of the performance diagnosis mechanism is the Root Cause Analysis (RCA) module. The RCA module is responsible to predict and localize faults and service degradations, so that in a next step the engineers and technicians can take decisions on how to improve the system or mitigate the possible faults. The RCA module uses diverse information including correlated network/service events and E2E service graphs. By correlated network or applications events, we mean events generated by different sources that can be related e.g. in

temporal or spatial way. For example, events from neighbouring nodes or events from the same source and subsequent time slots. In addition, service graphs are used as additional knowledge for the RCA algorithms in order to correlate nodes or link along a network path.
Inside the RCA resides the RCA algorithms, which are presented in more details in section 3.4.

Optimization generations
The last step of the performance diagnosis mechanism is the generation of network or service optimisation actions, which are in practise the steps and configurations that can be realised for the mitigation of possible faults and service degradations.
During this step, the network and service profiling results are taken into consideration. In 5G EVE the development of performance diagnosis mechanism will be completed with the development of a set of suggestion to the verticals. The next step would be the automatic triggering of measures at the network entities; it is out of the scope of the project, but is included in the figure for completeness. The same applies to the 3rd version of the Performance Diagnosis report.
The position of performance diagnosis mechanisms in the 5G EVE platform is depicted in Figure 27.**Error! Reference source not found.**

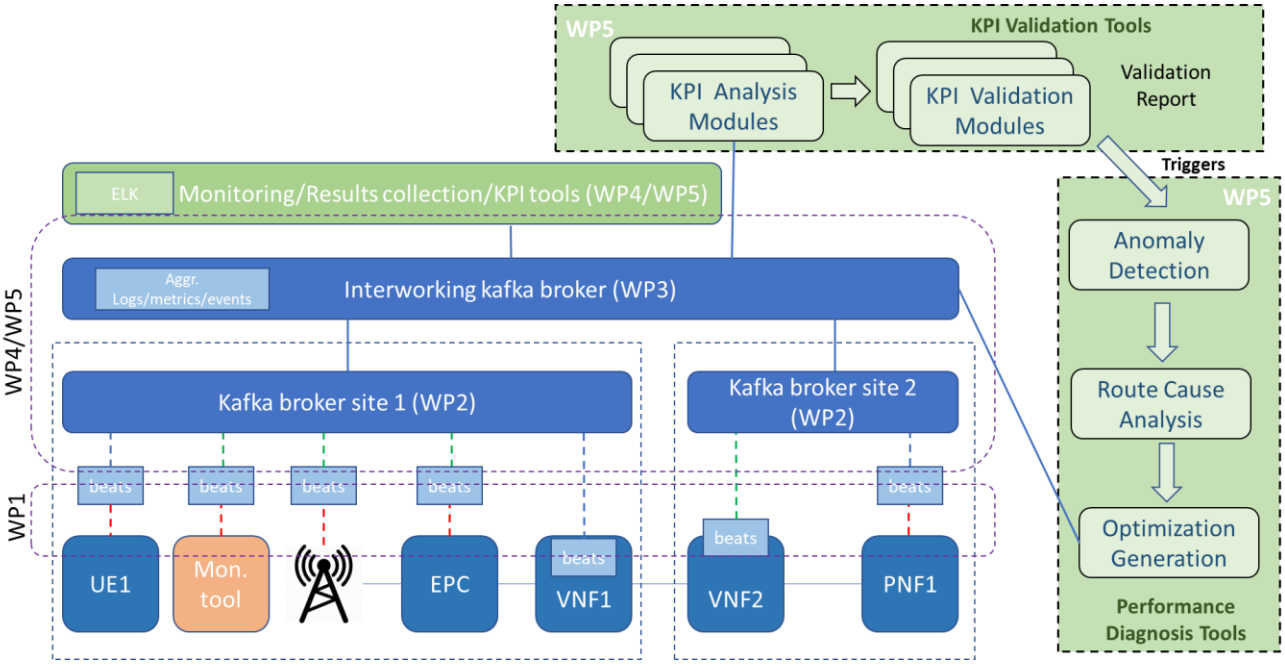


Figure 27: Performance diagnosis in 5G EVE

3.2 Performance diagnosis architecture

The approach of the performance diagnosis mechanism adopted in 5G EVE is described in the previous section, while in this section the general architecture of the performance diagnosis is introduced.
Error! Reference source not found.illustrates the general performance diagnosis architecture. According to this architecture, metrics can be collected from various network elements (left side of **Error! Reference source not found.**) including:

- Services and applications that logs information in the Server or even in the UE equipment.
- Network elements that log information.
- Data storage/processing elements located in the cloud that stores information on the Cloud.
- Network monitoring tools or Cloud related monitoring tools which collects metrics using agents.
- Active or passive probes located in the network responsible to collect metrics

The aforementioned elements then store the collected data/metrics in the Data Storage. In case of logging elements, a Data Ingestion Engine is used to retrieve the data and store them in the Data Storage. In case of agents, data shippers are used to store the collected metrics to the Data Storage (5G EVE D3.3 [5]).
The next step which **Error! Reference source not found.**realises the Performance Diagnosis mechanism performs data correlation, analysis and diagnosis. It includes the Data Correlation which is responsible to correlate data coming from different sources. During the correlation process the E2E service graphs and results from the service profiling are used in order to provide meaningful correlation among the data. Then data analysis is realised based on statistical procedures, while the process of anomaly detection is responsible to identify any anomalies in the retrieved data. The final step of the performance diagnosis is the application of the Root Cause Analysis (RCA) methodologies and algorithms on the post-processed data. The outcome of the RCA module is the identification of the cause of the performance degradation or the anomaly. In addition, a Diagnostics GUI illustrates to the end user, the findings of the performance diagnosis mechanism.

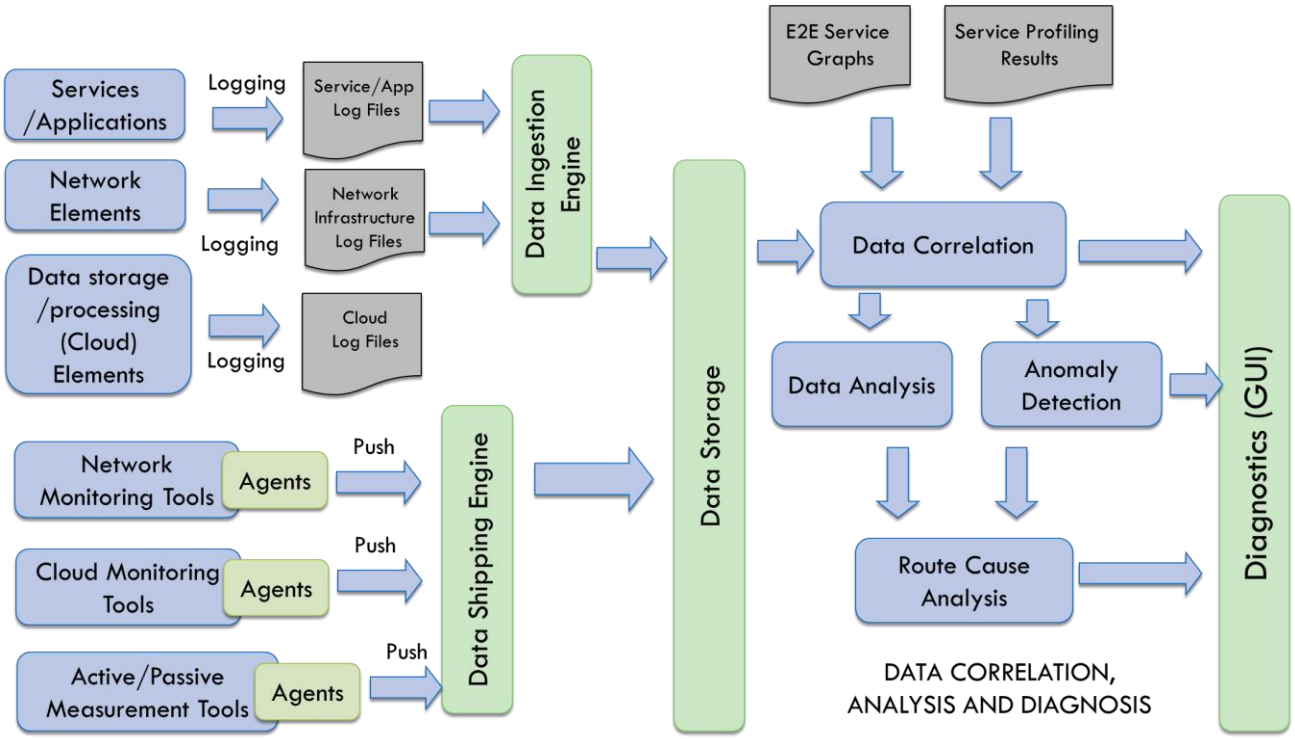


Figure 28: Performance diagnosis architecture

In 5G EVE the above architecture is adopted and followed for the development of the 5G EVE performance diagnosis module. The developed platform is illustrated in Figure 29.
In detailed, in 5G EVE platform the ELK Stack is used for the development. The Data Shipping Engine is integrated as a the Kafka Bus, the Data Ingestion Engine is developed using the Logstash, while the different agents acting also as data shippers to the Kafka Bus are realised using Beats. For the Data Storage, the ElasticSearch is used. The above components are parts of the Interworking Layer (IWL) of 5G EVE platform and are described in detail in D3.3 [5] (section 3.4 – Data Collection Manager).
The Data Correlation, Data Analysis, Anomaly Detection and RCA modules are realised using Python. The RCA algorithms of the RCA module because of their high importance of the performance diagnosis mechanisms are presented in detail in section 3.4.

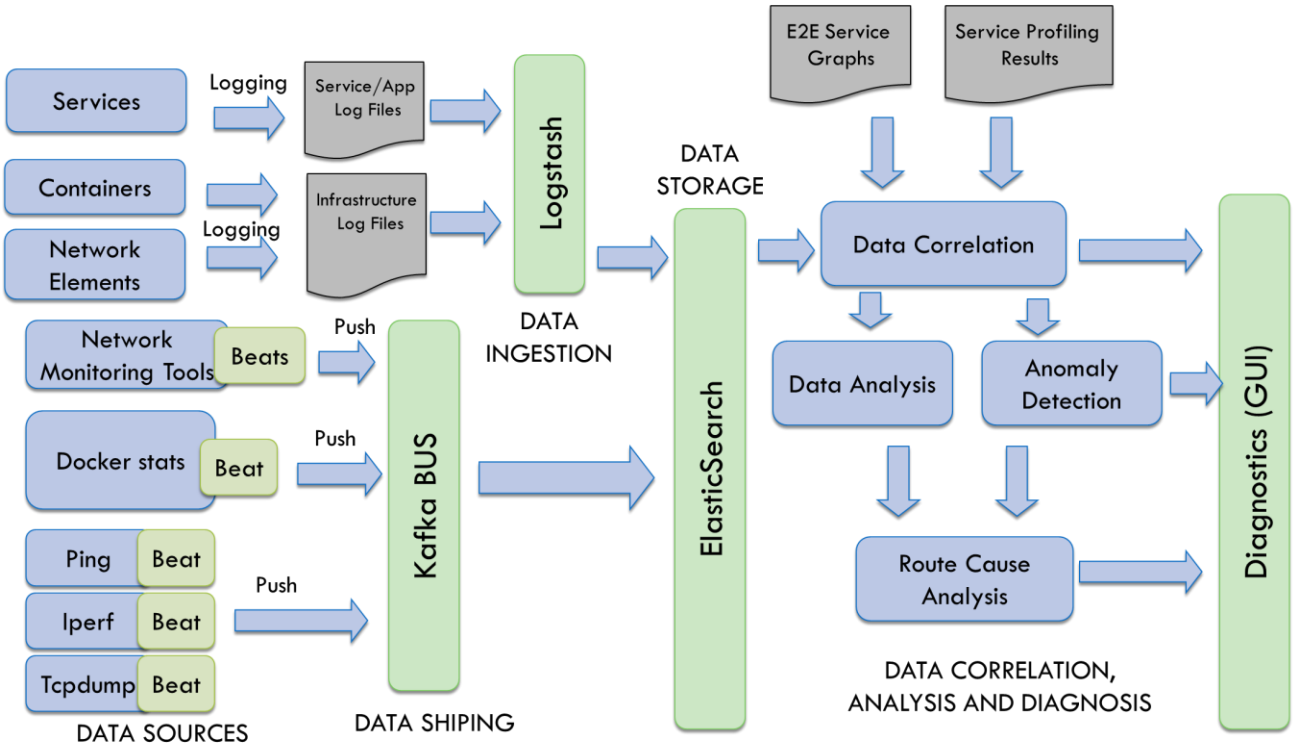


Figure 29: 5G EVE developments

3.3 Performance diagnosis workflows and APIs

The Performance Diagnosis (PD) of an experiment is an optional service requested from the experimenter and is planned to be integrated with the Portal in the next release of the platform. It is provided to the experimenter on a per experiment and per test case basis after a specific experiment has terminated. The Performance Diagnosis request process workflow including the interactions with other elements of the platform is shown in Figure 30 **Error! Reference source not found.**.

After a Performance Diagnosis is requested for a specific experiment, the PD module requests from the monitoring platform all the data generated during the execution of that experiment. This data will then be processed by the internal components of the PD module in order to establish the timeline and behaviour of the KPIs and perform a root cause analysis for any performance degradation. The monitoring module periodically checks the status of the report in order to notify the experimenter that the report is ready. After the diagnosis process has finished, the PD module notifies the RAV module about the completion of the report in order to embed the report in the composite report, consisting of both the RAV and the PD results, that will be provided to the Portal.

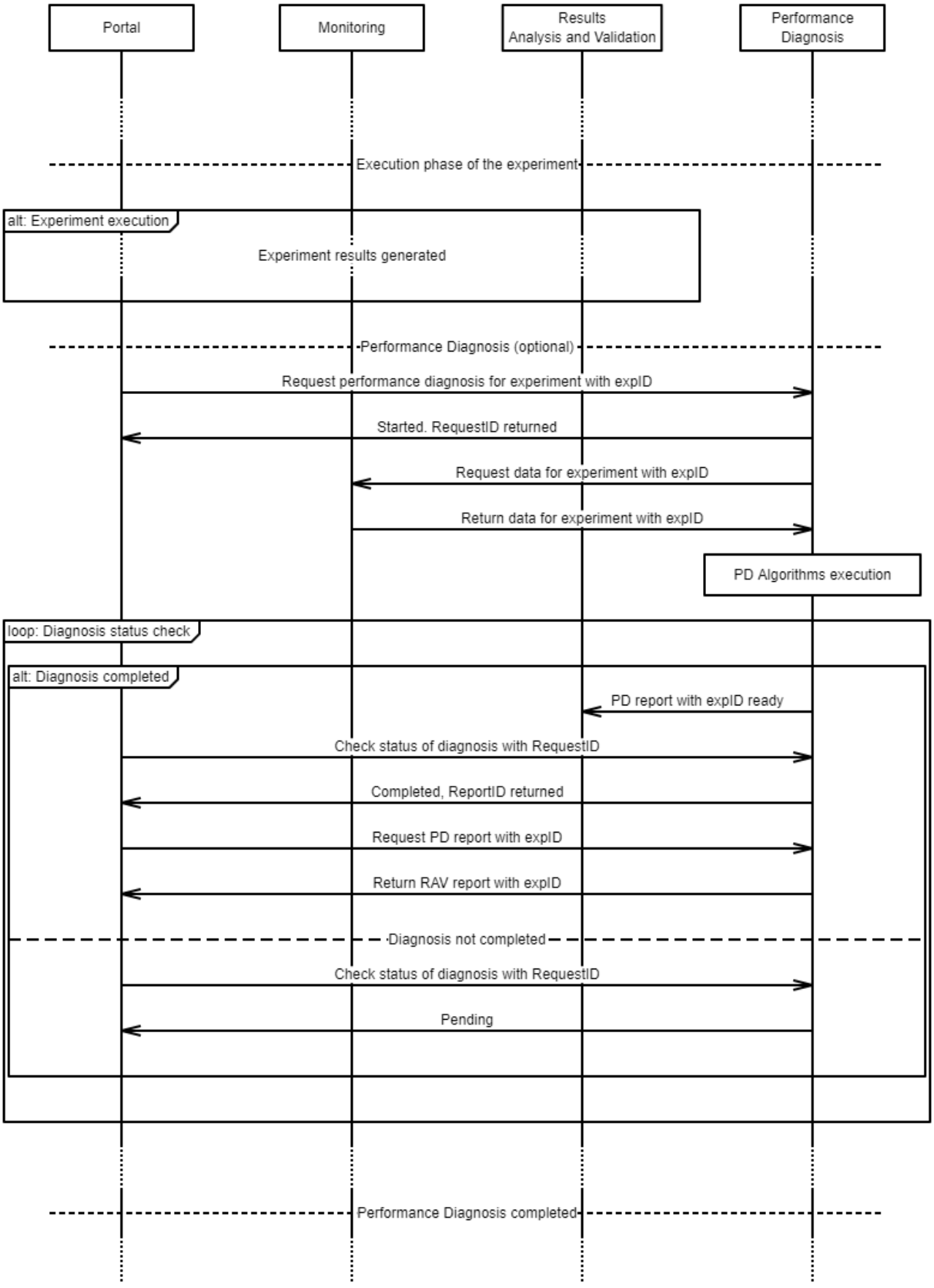


Figure 30: Performance Diagnosis request workflow

The Performance Diagnosis module implements the following API endpoints shown in Figure 31: Performance Diagnosis module APIFigure 31**Error! Reference source not found.**. The interactions between the module and the other components is implemented using a REST API[2].

Figure 31: Performance Diagnosis module API

The actions exposed via the REST API respond to a specific scope of the targeted resources and actions:

- *POST /experiment/{expID)/{executionID}/perfDiagnosis*: a performance diagnosis is requested for the specific execution of the experiment with the provided identifier.
- *GET /experiment/{expID)/{executionID}/perfDiagnosis*: the performance diagnosis report, if completed, for the execution of the experiment with the provided identifier is returned.
- *GET /experiment/{expID)/{executionID}/perfDiagnosis/status*: the status of a requested performance diagnosis report for the specific execution is returned
- *DELETE /experiment/{expID)/{executionID}/perfDiagnosis*: a generated report is deleted

When the Performance Diagnosis module receives a request, it sends a query to the monitoring platform for all the metrics and logs generated during that experiment. Upon receiving this data, the analysis algorithms are executed, and a report is generated.

3.4 Root Cause Analysis (RCA) algorithms

The process of Root-cause Analysis (RCA) has been used in numerous areas and is usually concerned with finding the root causes of events with safety, health, environmental, quality, reliability, production and performance impacts. RCA is a tool to help the user determine what and how something occurred, but also why something went wrong [2]. Having determined these three things, an effective recommendation can then be given about how to stop the event from occurring again.

The RCA process is usually divided into four major steps [2]:

1. Data collection:
   The first step of RCA will always be to gather the data necessary to localize a fault. This could range from asking every employee involved in a workplace incident to describe their experience, to monitoring the system resources of a server. From a cloud management point of view, one of the big challenges with data collection is the decision from where the data should be collected. If data measurements are collected from the wrong place, you will miss the essential data needed. However, if you instead try to collect all the data, you run the risk of losing the interesting part of the data in all the noise. In addition to this, the overhead of processing a lot of unnecessary data would be unwanted. Thus, a good balance needs to be found.
2. Casual factor charting/Prediction:

[2] https://github.com/5GEVE/performance-diagnosis

A causal factor chart is more or less a simple sequence diagram depicting the actions that led to an occurrence. In a computer related scenario this might be slightly harder to define and can be linked to a prediction algorithm able to detect faults when or before they occur. As such the predictions in a cloud management system should occur in real time and preferably make predictions of the future state of the system.

3. Root cause identification:
   When the sequence of events leading to an occurrence have been determined, they in turn can be used to determine the actual root cause of the occurrence. For a cloud management system this means finding the root-cause of any prediction that indicates a fault in the system.
4. Recommendation generation and implementation:
   When the root cause of an occurrence has been determined, recommendations on how to minimize or completely remove the cause of the occurrence can be made. The recommendation and method vary with the system that has been analysed. In a cloud management system, the optimal solution would be to point to any possible system metrics or components that could facilitate the maintenance of the system. In 5G EVE we are limited to the recommendation generation phases, since the actual implementation of the optimisation actions are out of the scope of the project.

An often-used alternative approach to complex analytical models is to design and implement models based on statistical learning **Error! Reference source not found.**. This allows for models that learn the system behaviour from observation of system metrics and once implemented can make predictions of the system behaviour based on future observations. The downside is that a large amount of data containing observations need to be gathered, however the upside of this is that no knowledge about the system and inter component interactions is needed.
Another alternative is that machine learning builds models based on observational data and the user does not need to understand the underlying complexity of the system, and can thus provide accurate predictions of future observations. Another major strength found in some machine learning algorithms is that the model is able to learn the topological properties of the observational data [3], which is something that we leverage in this project in order to provide fault localization in the system.
In the project, the focus was both on the design and development of statistical learning approach as well as developing solutions that explore the possibilities of a machine learning approach. Two RCA algorithms are designed and developed in the scope of the 5G EVE project. The first is using statistical learning and its characterised by low complexity and high speed in finding the root cause. The second algorithm is using machine learning approaches (Self Organizing Maps - SOMs) and it is characterised by medium complexity and the requirement of a long training data set.

3.4.1 RCA algorithm based on adjacency list

The first algorithm developed into the Performance Diagnosis module is based on adjacency list [3]. The physical topology information is available as an n-node undirected graph represented as an adjacency list, adjList [4]. Specifically, all the nodes of the managed network are numbered from 1 to n. The adjacency list (adjList) is an array (size n) of linked lists where index i of the array contains the linked list of all the nodes directly connected by a network link to node i.

The algorithm has two parts:

- Part A determines the status of individual elements in the network. The various real statuses can be Up, Down and Unknown. An element with real status Down is the root cause for itself.
- Part B assigns a particular Unknown node to a set of Down node(s) where the Down contour is the root cause(s) for the particular Unknown element. That is, given any Unknown node ‘a’, it returns the list of Down nodes {b | a depends on b is true}. Conversely, for a particular Down node it returns a list of Unknown nodes where the Down node has caused all those nodes to be Unknown

The flowchart of the algorithm is illustrated in Figure 32**Error! Reference source not found.**.

3.4.2 RCA algorithm based on Self Organizing Maps

The second algorithm developed into the Performance Diagnosis module is using Self Organizing Maps (SOMs)**Error! Reference source not found.**. The SOM approach is an unsupervised learning technique where the maps represent high-dimensionality data in a low dimensionality view, while preserving the topological properties of the data. Traditionally, a SOM is represented as a rectangular or hexagonal grid in two or three dimensions.

The algorithm is fed with various metrics in the form of input vectors. Then these vectors are used as training set of the SOM and for the detection and localization of the SLA violations.

The output of the algorithm is a vector containing the nodes that are most likely the root cause of the SLA violations, ranked by starting from the most probable one.

The algorithm is realised in four steps:

1. Required data collection/storage:
   The algorithm collect the following data to be fed as the input vectors: Latency, Throughput, Memory[kB], CPU[%], I/O[per sec.], Block I/O[per sec.], Network[per sec.]
2. Training:
   - Initialize the weights of each node in the SOM (using random values)
   - Present a randomly chosen vector from the training set to the SOM
   - Find the Best Matching Unit (BMU) node in the entire SOM that most closely resembles the input vector (Euclidean Distance between input vector and weigh vectors)
   - Find each node that belongs to the neighborhood of the BMU; the neighborhood is defined by the neighborhood function (often represented as the radial distance between the coordinates of two nodes)
   - For each node, update the weights so that the nodes more closely resemble the input vector
3. Anomaly detection:
   - Present a new sample X to the trained SOM
   - Compare X to the weight vector of each node in the trained map. The best matching node Mi to X is found using the Euclidean distance
   - Calculate the neighborhood size S of the node Mi that X is mapped to. S(Mi) is calculated as the sum of the Manhattan distance D between Mi and the nodes adjacent to it MT;MB;ML;MR)
   - Compare S to a pre-set threshold TS. If S _ TS then X is predicted as faulty(SLA violation), otherwise X is predicted as healthy(non-SLA violation).
4. Fault localization
   - Present a new sample X of an unhealthy node to the trained SOM
   - Locate the N nearest healthy nodes to X using Manhattan distance
   - Calculate the dissimilarity vector DS_{tot} for X

   The dissimilarity vector can then be used to interpret the nature of the fault present in the sample, since each element in the dissimilarity vector DS_{tot} corresponds to a feature xi in X. By sorting DStot in descending order, one receives a ranking list where the top ranked results are more likely to be the cause of the detected fault.

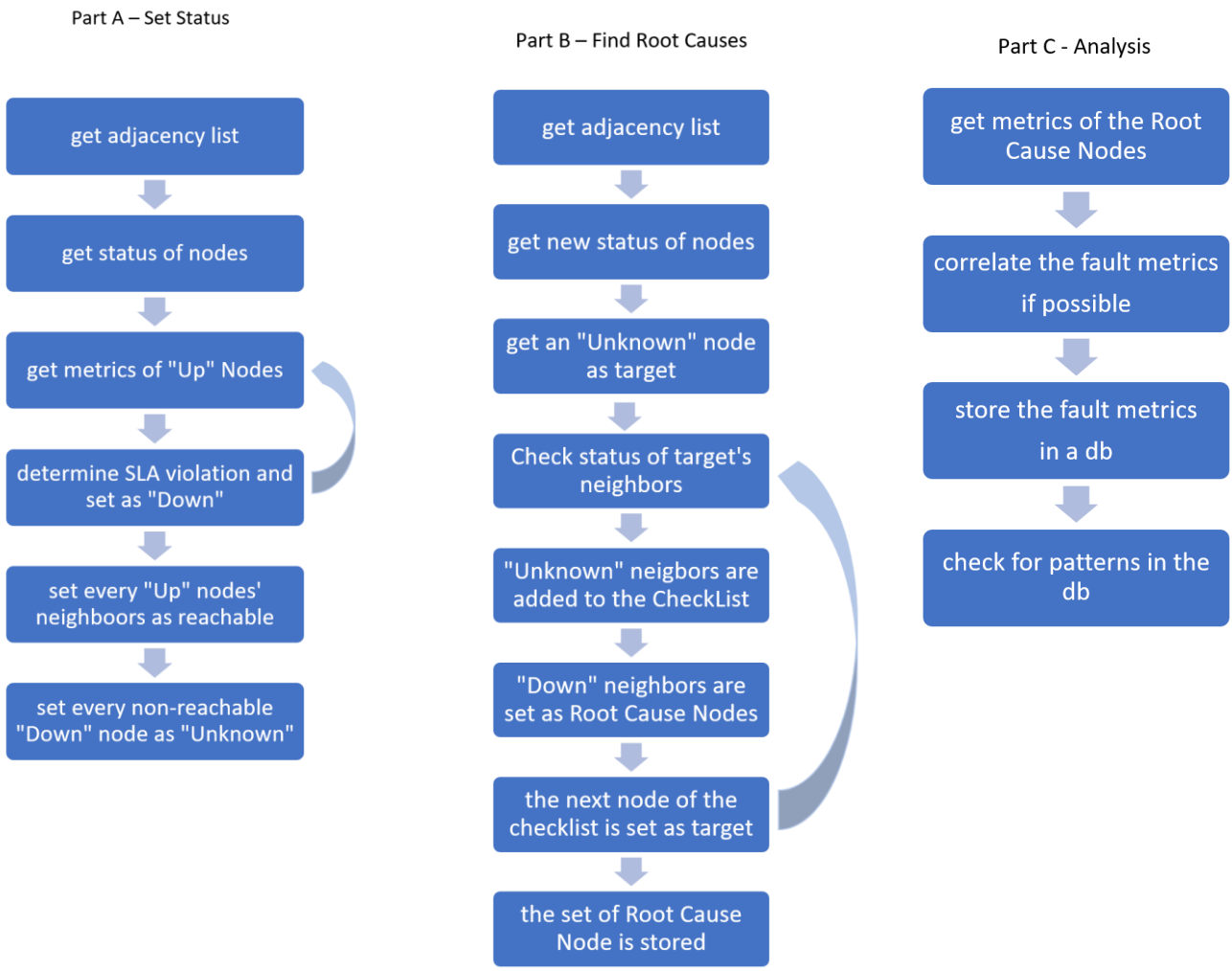


Figure 32: RCA algorithm based on adjacency list

3.5 Performance diagnosis results

At the time this document was written, the development and testing of the first RCA algorithm (based on adjacency list) was completed, while the second RCA algorithm (based on Self-Organizing Maps) was under development. Therefore, the following paragraphs presents the testing and evaluation results of the first RCA algorithm.

3.5.1 Experimental Scenarios

To test the algorithm's functionality a certain scheme was implemented, and various scenarios were configured, based on that scheme. The tree topology was selected as the initial topology used to validate the RCA algorithm since this kind of topology are common in the access networks. The network topology used in the experimentation is shown in Figure 33.

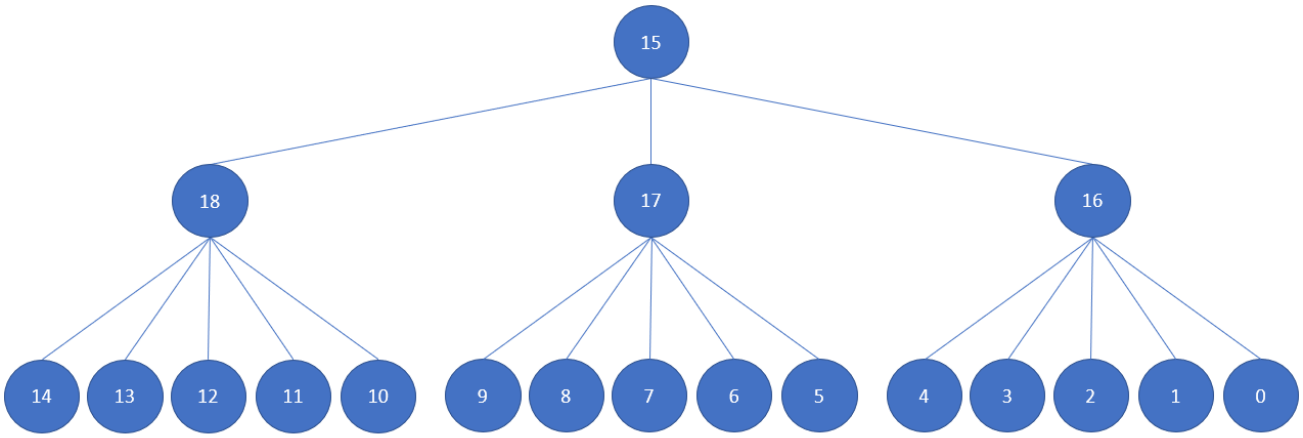


Figure 33: The topology of the testing scheme

Nodes 15-18 are routers and nodes 0-14 are hosts. Hosts send requests for a dummy calculation to other hosts, usually to a host belonging in a different cluster, and they get an according answer with the calculated result, producing the "request latency" metric. CPU and RAM overload is injected to the routers and negative impact of that fault injection on the request latency metric is expected. An example scenario is shown in Figure 34.

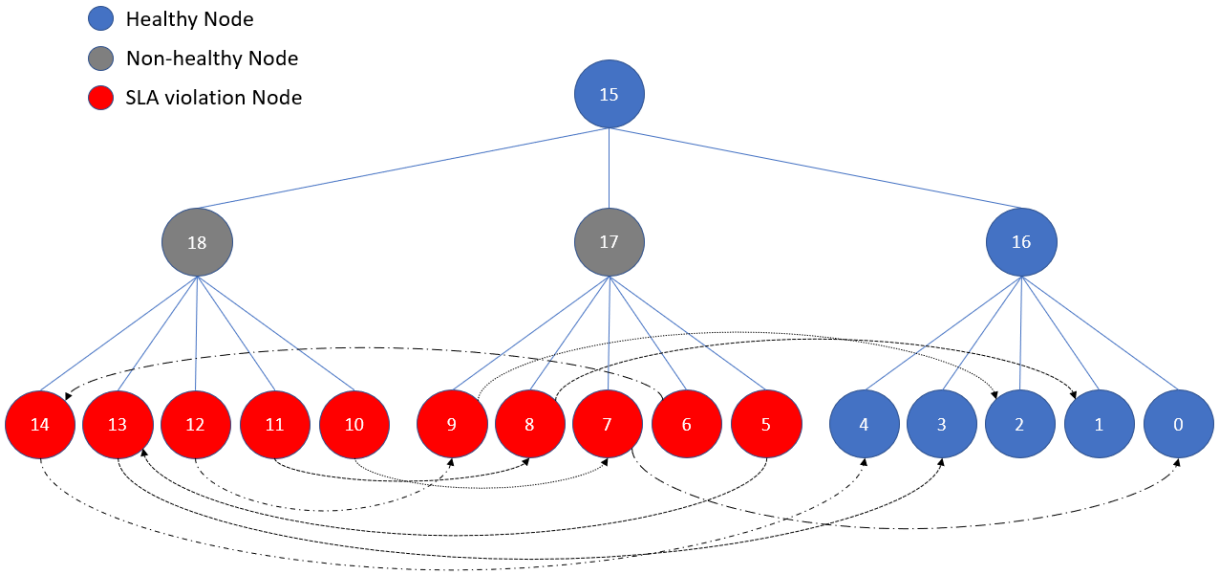


Figure 34: An example of the implemented test scenarios

For the scenario above, the algorithm successfully detected Node 18 as the root cause for the SLA violations of Nodes 10-14 and Node 17 as the root cause for the SLA violations of Nodes 5-9. The dotted lines show the requests sent by the hosts.

3.5.2 Testing Procedure

The testing procedure took place using the algorithms continuous (automatic) mode. A testing scenario involving fault injection in one of the router nodes 16-18 at a time is implemented. Heavy CPU and RAM loads are added to a different router, every 15 mins. The diagnostic algorithm fetches metrics from the system every 10". Then it runs and stores the results for that metrics frame. After a user-defined time period, the results are collected and analysed.

3.5.3 Results

The results of the above procedure running for approx. 5 hours are shown in Figure 35**.**

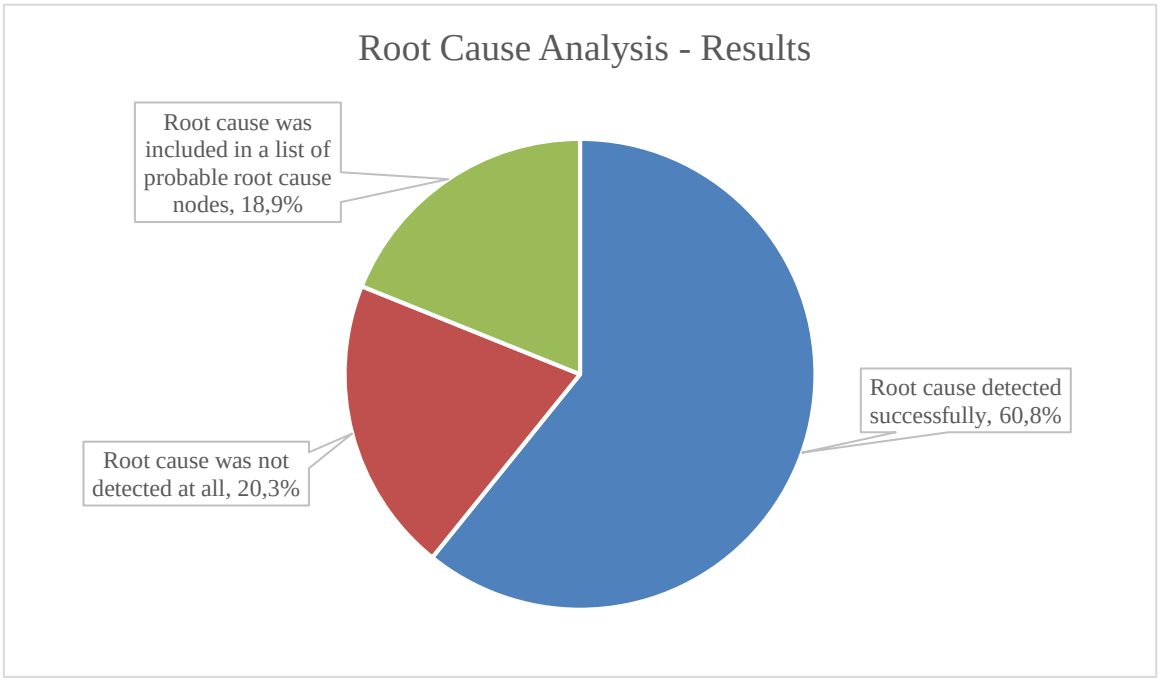


Figure 35: Test results chart

As expected, the system overload added to the router nodes produced SLA violations. The algorithm detected the root cause node in 60.3% of the violations. In the rest of the cases, two possible events account for the algorithm's failure. Firstly, it is possible that the NMS (Node 0) was "blocked" by system overload or traffic on Node 16, thus, detecting Node 17 or Node 18 as the root cause was unachievable. The second possible event is that an SLA violation was taking place on the request receiver, but it was caused from system overload on the cluster router of the request sender. Since the cluster router of the request receiver was healthy, the algorithm could not detect the exact root cause. In 18.9% of the attempts to locate the exact root cause, the latter was included in a list of probable root cause nodes. In the remaining 20.3%, the algorithm was unable to detect the presence of a root cause node and was only able to acknowledge the SLA violation.

4 Conclusions and next steps

In this document the reporting framework and the performance diagnosis mechanism of the 5G EVE testing and validation platform are presented.

Regarding the reporting framework, the adopted methodology, the architecture and the workflow and APIs are presented, focusing on the components that mainly interacts with the reporting framework, which are the Experiment Execution Manager and the Results Analysis and Validation components. In addition, indicative reporting results from the application of the reporting methodology on different use cases are presented.

Regarding the performance diagnosis mechanism, the general approach adopted in 5G EVE for the design of the performance diagnosis mechanism is presented. In this framework, the various steps of a performance diagnosis methodology are presented, providing details for each step. Then, the workflow and APIs are presented in order to stress the interaction with the other 5G EVE components. The design and development of performance diagnosis and root cause analysis algorithms are the heart of the mechanism. Therefore, the adopted algorithms are presented in detail as well. Finally, initial results from the application of the performance diagnosis on emulated networks are presented and analysed.

The next steps toward the completion of the reporting framework are the specialisation of the adopted methodology for each KPI, in order to clearly meet the requirements and the special needs of each KPIs.

The next steps toward the completion of the performance diagnosis mechanism are the application of the mechanism to the actual 5G EVE infrastructure and use cases.

References

[1] 5G EVE D5.2, “Model-based testing framework”, December 2019

[2] 5G PPP (2019). Validating 5G Technology Performance [White Paper]. https://5g-ppp.eu/wp-content/uploads/2019/06/TMV-White-Paper-V1.0-03062019.pdfJames J Rooney and Lee N Vanden Heuvel. “Root cause analysis for beginners. Quality progress,” 37(7):45–56, 2004.

[3] Bhattacharya, K., Rani, N. U., Gonsalves, T. A., & Murthy, H. A. (2005). An efficient algorithm for root cause analysis. In *Proc of the 11th National Communications Conference* (pp. 447-451).

[4] Tremblay and Sorenson, “An introduction to data structures with applications”

[5] 5G EVE D3.3. “First implementation of the interworking reference model”, October 2019